UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
TELEMÁTICA


PROYECTO FINAL DE CARRERA

# MÓDULO DE DISTRIBUCIÓN Y PRE-CLASIFICACIÓN DE CONTENIDOS PARA UNA PLATAFORMA DE INTERCEPTACIÓN LEGAL DE COMUNICACIONES

AUTOR: RUBÉN MENA ESCRIBANO
TUTOR: MANUEL URUEÑA PASCUAL

20 de Julio de 2011

# Agradecimientos

Quisiera expresar mi agradecimiento a todas aquellas personas que me han apoyado en este largo camino educativo, todos aquellos que siempre me han animado a continuar y me han dado fuerzas para llegar hasta aquí.

En primer lugar quisiera agradecerles a mis padres el continuo apoyo que me han ofrecido desde que era pequeño, siempre estando a mi lado y empujándome a conseguir mis objetivos, no me cabe la menor duda que sin ellos hoy no estaría escribiendo estas líneas, muchas gracias de corazón a Miguel y María Teresa.

No quisiera dejar de agradecerle a mi primo Jorge toda la ayuda prestada y su continuo interés, ya que siempre ha estado ahí para todo lo que he necesitado, no tan sólo en el ámbito académico sino personal, espero tenerte siempre cerca porque eres una de las mejores personas que conozco.

Por otro lado, me gustaría agradecerle a todos los profesores y compañeros que he tenido y que de una forma u otra han contribuido a mi formación, la ayuda prestada a lo largo de la carrera. En especial hacer una mención a mi tutor de proyecto Manuel Urueña, que es un profesor fantástico y que tanto en su etapa como profesor y tutor de proyecto siempre ha sido muy educado, simpático y dispuesto a ayudarme en todo lo que he necesitado. También deseo dedicar unas palabras a Carlos Gacimartín por su ayuda y disponibilidad en la realización del proyecto.

Por último, pero no menos importante darle las gracias a mis amigos por haber confiado siempre en mi y haber sabido entender los grandes sacrificios que muchas veces he tenido que hacer por dedicación a mis estudios. Siempre habéis estado apoyándome y dándome ánimos.

Todos vosotros que habéis pasado por mi vida habéis contribuido de una forma u otra en mi formación académica y como persona. Todas las vivencias experimentadas con vosotros me han convertido en lo que hoy en día soy, de modo que muchas gracias a todos.

Un cordial saludo de vuestro familiar, alumno y amigo, Rubén Mena Escribano.

# Índice general

# Lista de Figuras

# Lista de Tablas

# Resumen

El presente proyecto documenta el proceso que se ha llevado a cabo en el diseño e implementación de un módulo software de distribución y pre-clasificación de contenidos de red, basado en una arquitectura de Plugins intercomunicada mediante *Webservices*, que recibe el nombre de PluginManager.

Siguiendo la línea de investigación y desarrollo del proyecto *'INDECT'* se pretende que éste módulo pueda ser integrado con la herramienta de análisis forense de tráfico de red *'Xplico'* para el despliegue de un sistema de interceptación legal de comunicaciones basado en tráfico de red.

La pre-clasificación de contenidos llevada a cabo está relacionada tanto con el procesado de contenidos únicos de internet como de aquellos que contribuyen en la realización de actividades ilícitas en el mundo real, que emplean internet como medio de comunicación. Esta información de pre-clasificación servirá para facilitar al operario del sistema, que será un mimbro del cuerpo de seguridad de un País Europeo, a llevar a cabo una clasificación final del contenido que determinará si dicho contenido es relevante en una investigación o si por el contrario puede descartarse.

Capítulo 1

# INTRODUCCIÓN

## 1.1. El proyecto INDECT

- **Presentación**

  El Proyecto INDECT (Intelligent information system supporting observation, searching and detection for security of citizens in urban environment) es un proyecto de investigación financiado por la Unión Europea, en el que trabaja la Policía y universidades de diferentes países de la UE, que consiste en el desarrollo de nuevos algoritmos y métodos avanzados destinados a combatir el terrorismo y otras actividades delictivas que se llevan a cabo tanto dentro como fuera de Internet y afectan a la seguridad de los ciudadanos.

  Cabe destacar la existencia de dos formas de delinquir en la red, aquella que engloba delitos que únicamente se dan en internet, como podría ser el intercambio de contenidos en las redes P2P (*peer to peer*), y aquellos que pertenecen al mundo real pero que utilizan internet como medio de comunicación, de éste último cabe destacar el tráfico de drogas, armas, terrorismo, etc.

  De modo que, en resumidas cuentas, el Proyecto INDECT trata de combatir todo tipo de crímenes que utilicen Internet de un modo u otro para llevar a cabo actividades ilícitas.


- **Orígenes y participantes**

  INDECT fue iniciado por la *'Plataforma para la Seguridad Nacional de Polonia'*, cuya propuesta fue presentada por el *'Consorcio Internacional Pan-Europea'* conformada por 17 socios, liderados por la Universidad AGH de Ciencia y Tecnología (Cracovia, Polonia), bajo la supervisión del profesor Andrzej Dziech, el coordinador del Proyecto. El consorcio se encuentra formado por 11 universidades conocidas, entre las que se encuentra la Universidad Carlos III de Madrid. Además cabe destacar la participación de la Policía de Irlanda del Norte y el Cuartel General de la Policía de Polonia.

- **Objetivos**

  El objetivo principal del proyecto es el desarrollo de una infraestructura que lleve a cabo la monitorización de un entorno urbano a través de un procesamiento inteligente de la información capturada para detectar automáticamente amenazas y comportamientos anómalos de los usuarios.

  Los principales objetivos del proyecto INDECT son:

  - Desarrollo de una plataforma para el registro y el intercambio de contenidos multimedia, procesamiento inteligible de la información y la detección automática de amenazas, y el reconocimiento de la conducta criminal o la violencia.
  - Desarrollo de un prototipo de un sistema integrado que sirva como red social centrada en el apoyo a las actividades operativas de los policías, proporcionando técnicas y herramientas para la observación de varios objetos móviles.
  - Desarrollo de un nuevo tipo de motor de búsqueda que combina la búsqueda directa de imágenes y de vídeo basado en los contenidos marcados por agua y el almacenamiento de los meta-datos en forma de marcas de agua digitales.

- **Aplicaciones**

  Resultará útil en investigaciones legales de ámbito confidencial, tanto para entornos virtuales como reales:

  - Pornografía infantil.
  - Tráfico de órganos humanos.
  - Promoción de símbolos totalitarios.
  - Propagación de redes de boots, virus, malware.
  - Terrorismo
  - Vandalismo
  - Robos

  Además también sirve para la detección de amenazas tales como:

  - Abandono de equipaje, lo cual podría resultar ser una amenaza de atentado.
  - Gente presente en los carriles de las vías de servicio comunitario.
  - Fuegos.

- **Relación de INDECT con el presente Proyecto Final de Carrera**

  Dentro de INDECT el objetivo de la Universidad Carlos III es la colaboración en el desarrollo de herramientas para la interceptación legal de las comunicaciones.

## 1.2. La herramienta 'Xplico'

- **Presentación**

  Xplico es una herramienta de análisis forense de tráfico de red, que tiene por objeto decodificar el tráfico de red capturado mostrando datos útiles para el usuario a través de un interfaz gráfico desarrollado en CakePHP.

  Existen multitud de herramientas de análisis de red, como por ejemplo, Wireshark, Tshark, TCPDump, etc... pero estas proporcionan información de tráfico de red de bajo nivel, es decir, aportan información muy detallada de los contenidos capturados, mientras que en ocasiones puede darse la situación en que se quiera analizar el tráfico de una forma más general. Precisamente lo que hace Xplico es tomar el tráfico capturado y reconstruir los protocolos de aplicaciones de datos desarrollando un registro de contenidos con sus correspondientes características. Esta información generada es presentada en un interfaz de usuario más amigable (que la de los comunes analizadores citados anteriormente), para que el usuario pueda interpretar los datos de una forma más sencilla y visual.

- **Origen y participantes**

  Herramienta desarrollada por *'Gianluca Costa & Andrea de Franceschi'* a inicios del año 2007 hasta la fecha. Actualmente se encuentran en la versión 0.6.2 y continua en desarrollo. Este software se distribuye bajo una licencia de código abierto, lo cual quiere decir que cualquiera que esté interesado en trabajar en su desarrollo no tiene más que ponerse en contacto con los desarrolladores y solicitarles información a cerca de la herramienta. Si ahondamos en su página oficial (http://www.xplico.org) podemos ver que se encuentra bastante bien documentada, con la finalidad de que la gente se anime a la creación de pequeños módulos que ayuden al crecimiento de la aplicación.

- **Objetivos**

  El objetivo principal de Xplico es la elaboración de una herramienta capaz de reconocer la gran mayoría de los protocolos que actualmente se encuentran en auge en Internet, independientemente del puerto que utilicen, y reconstruir los protocolos de aplicación de datos que los llevan. Por ejemplo, para hacernos una idea, imaginemos que Xplico a partir de un fichero de captura de tráfico de red dónde se centra en el análisis de los protocolos de nivel

de aplicación coge la información en crudo capturada, deberá ser capaz de reconstruir todas las páginas y contenidos web que transporta el protocolo HTTP, y los correos electrónicos (cabeceras, texto del cuerpo, elementos adjuntos, etc) que transportan los protocolos POP, IMAP, SMTP, y así con todos los demás protocolos de nivel de aplicación disponibles. Estos contenidos reconstruidos a partir de la información en crudo proporcionada por un simple sniffer de tráfico será almacenada en el disco indicado y generará información de registro tal como su localización, tipo de contenido y otros parámetros de interés con los que poder trabajar.

- **Características**

  - Multitud de protocolos soportados

    Xplico es ambicioso y pretende disponer dentro de poco de soporte para la gran mayoría de los protocolos más populares en la actualidad. Muchos de ellos ya se encuentran desarrollados al 100 %, pero muchos otros se encuentra aún en desarrollo. Estos módulos encargados del análisis de los datos en crudo capturados y su posterior reconstrucción reciben el nombre de *'Disectors'*, y pueden consultarse cual es su actual estado en la Figura 1.1, la cual ha sido extraída de la página oficial.

| Dissector | Status | Note | Dissector | Status | Note |
|---|---|---|---|---|---|
| ARP | 90% | — | PJL | 90% | — |
| Radiotap | 90% | — | NNTP | 95% | — |
| Ethernet | 100% | — | MSN | 60% | v1 beta |
| PPP | 90% | — | IRC | 85% | — |
| VLAN | 95% | — | YAHOO | 0% | — |
| L2TP | 70% | — | GTALK | 0% | — |
| IPv4 | 98% | — | EMULE | 0% | — |
| IPv6 | 98% | — | SSL/TLS | 0% | with keys |
| TCP | 95% | — | IPsec | 0% | with keys |
| UDP | 100% | — | 802.11 | 60% | no encryp. |
| DNS | 80% | — | LLC | 60% | — |
| HTTP | 100% | — | MMSE | 95% | over HTTP |
| SMTP | 95% | — | Linux cooked | 95% | SLL |
| POP | 95% | — | TFTP | 90% | — |
| IMAP | 95% | — | SNOOP | 100% | Format |
| SIP | 80% | — | PPPoE | 90% | — |
| RTP | 70% | — | Telnet | 90% | — |
| RTCP | 60% | — | WebMail | 90% | — |
| SDP | 70% | — | Paltalk Exp. | 60% | — |
| FB chat | 90% | — | Paltalk | 90% | — |
| FTP | 90% | — | NetBIOS | 5% | Ses. Mes. |
| IPP | 90% | — | SMB | 0% | — |

Figura 1.1: Protocolos soportados por Xplico (Disectors).

Referencias http://www.xplico.org/status

- Soporte PIPI (Port Independent Protocol Identification)

  Como su nombre indica Xplico es capaz de identificar protocolos independientemente del protocolo que utilicen.

- Multithreading

  Mecanismo conseguido gracias al aprovechamiento de la arquitectura multi-núcleo de las CPUs y al soporte multi-hilo por núcleo.

- Registro de contenidos analizados

  Una vez han sido reconstruidos los contenidos de red capturados se registrarán en bases de datos del tipo SQLite o MySQL. En la Figura 1.2 y Figura 1.3 se pueden apreciar los diferentes campos de que constan los registros de contenidos.

  Además por cada uno de los datos reconstruidos por Xplico se asocia un archivo XML que contiene la información técnica de bajo nivel de los flujos y el fichero de captura *'pcap'* que contiene los datos mostrados.

- Soporte para IPv4 and IPv6

- Modularidad

  La interfaz de entrada, los decodificadores de protocolos (Disectors) y el interfaz de salida (Dispatcher) se han diseñado como módulos independientes, para que sea sencillo reemplazarlos o añadir otros nuevos.

- **Modos de Funcionamiento**

  Puede funcionar de dos formas diferentes:

  - Online (análisis en tiempo real)

    Incorpora en su estructura un *'Sniffer'*, mediante el cual puede capturar información de red en tiempo real, simplemente configurando los parámetros de escucha adecuados (interfaz, filtros, etc). Una vez finalizada la escucha, pasará al procesado del tráfico capturado.

  - Offline

    Dispone también de la opción de analizar tráfico de red fuera de linea, es decir, a partir de ficheros *'.pcap'* que fueron generados con anterioridad por un *'sniffer'* cualquiera (no necesariamente el de Xplico).

- **Interfaz gráfica proporcionado**

  Proporciona un *front-end* desarrollado en *CakePHP* donde mostrará la información del tráfico analizado. La apariencia actual del mismo se puede apreciar en la Figura 1.5.

- **Licencia GNU/GPL**

  Xplico está licenciado bajo la GNU/GPL *(General Public License)*, lo cual quiere decir que podemos trabajar con dicho software abiertamente, ampliándolo de manera sencilla. El software correspondiente se puede descargar desde su página oficial:

  http://www.xplico.org/download

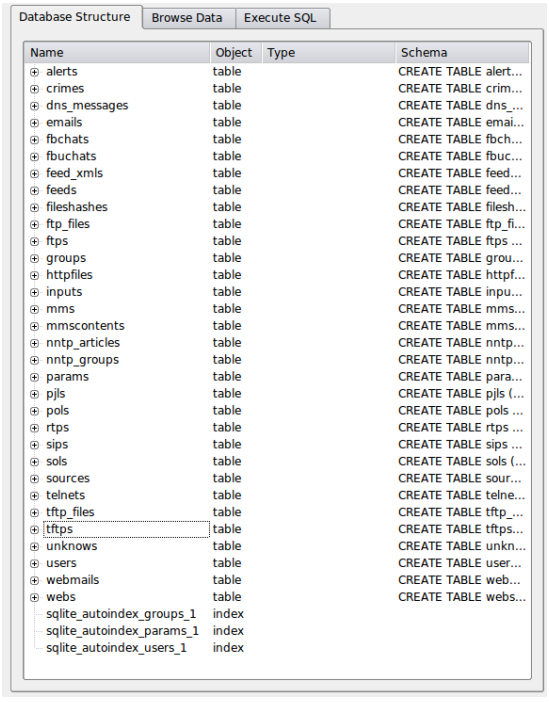


Figura 1.2: Aspecto de la Base de datos de la herramienta Xplico. Lista de Tablas.

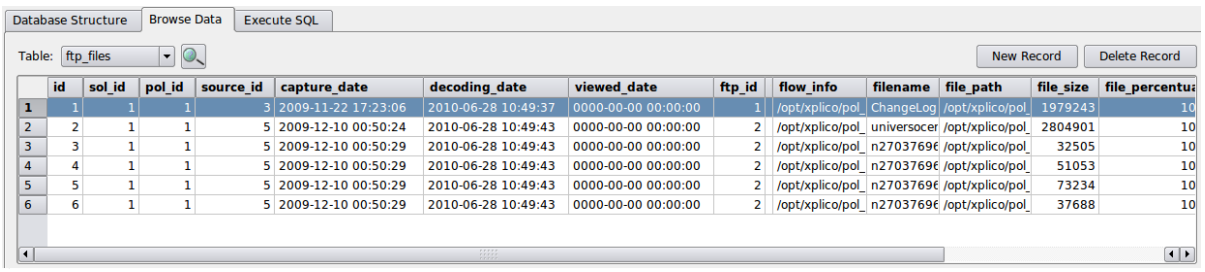


Figura 1.3: Tabla de sesiones TFTP de Xplico.

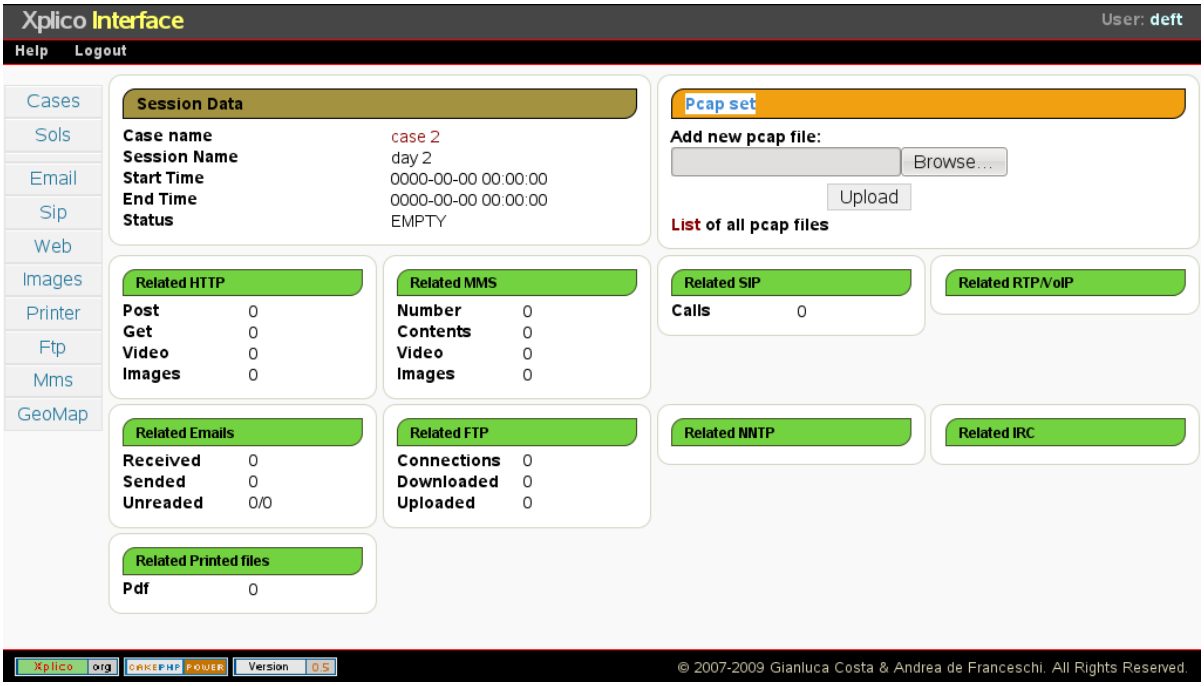


Figura 1.4: Interfaz de usuario de la herramienta Xplico: índice de la sesión.

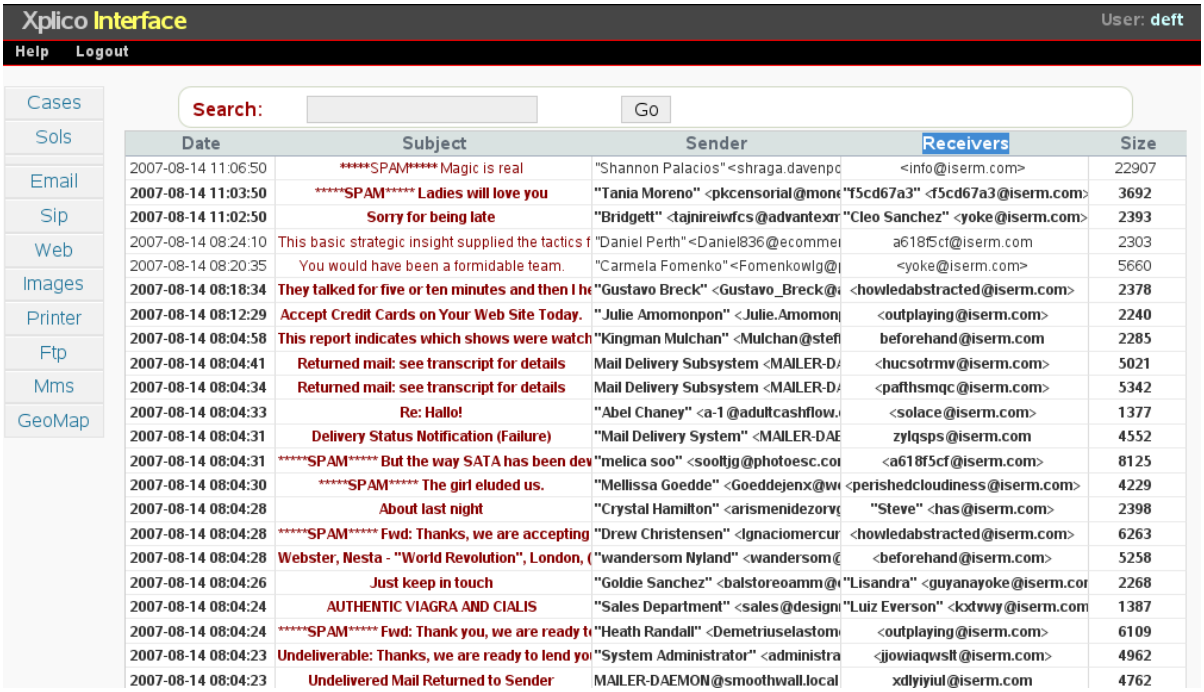


Figura 1.5: Interfaz de usuario de la herramienta Xplico: correo electrónico.

## 1.3. Motivación y objetivos del proyecto

Con el objeto de desplegar un *'Sistema de Interceptación Legal de Comunicaciones'* que pueda ser de utilidad a los Cuerpo de Seguridad Europeos, se expone en el presente Proyecto la elaboración de un software capaz de aprovechar la información proporcionada por un decodificador de tráfico de red para llevar a cabo el correspondiente procesado de contenidos extraídos y su posterior pre-clasificación mediante la asignación de prioridades.
La idea surgió a partir del conocimiento de la herramienta de análisis forense de tráfico de red *'Xplico'*, la cual como se ha comentado en la sección anterior, filtra, decodifica y muestra el tráfico de red capturado, generando una interfaz web de usuario dónde se representa el resultado del proceso. La idea es aprovechar esta información para poder clasificar los contenidos capturados y actualizar el interfaz web con el resultado obtenido.
De este modo, se consigue un sistema tal que ayude al personal de seguridad que lo utilice a discernir si los contenidos de red que han sido procesados son importantes y deberán ser examinados más detalladamente o si por el contrario no son relevantes y pueden ser desechados (por ejemplo, mensajes de spam).

La motivación que mueve a las Fuerzas de Seguridad a la utilización de este tipo de Sistemas de Investigación, viene determinado por la dificultad de controlar los delitos que se llevan a cabo en Internet, ya sea en ámbito exclusivo de internet o en su utilización como medio de comunicación en actividades ilícitas del mundo real. Disponer de un buen sistema de pre-clasificación de contenidos es fundamental ya que facilitará mucho la labor de clasificación final llevada a cabo por el personal de seguridad. Por lo tanto, la finalidad que se pretende conseguir es la elaboración de un Sistema de Interceptación Legal que permita ahorrar tiempo en la detección de contenido malicioso en la red y ahorrar recursos.

Para hacernos una idea aproximada de la situación descrita, se comentará a continuación la arquitectura global del sistema representada en la la Figura 1.6:

- Decodificador de tráfico de red Xplico

  Será quien analice el tráfico capturado y proporcione la información necesaria al Distribuidor de contenidos (PluginManager) para que éste pueda llevar a cabo el proceso de pre-clasificación. Esta información que Xplico le pasa al PluginManager será comentada en siguientes apartados, pero a grandes rasgos será: localización del contenido, nombre del

contenido, tipo del contenido, etc. Este modulo decodificador será la herramienta Xplico, de la cual ya hablamos en la sección anterior.

- Distribuidor de contenidos: 'Xplico PluginManager'

  Xplico PluginManager será el modulo software a diseñar e implementar en este Proyecto, el cual llevará a cabo los mecanismos necesarios para distribuir los contenidos proporcionados por Xplico a los módulos encargados de su procesamiento y etiquetado, es decir, a los Plugins. Llevará a cabo las altas y bajas de los diferentes Plugins de procesamiento, gestión de las diferentes políticas de distribución de contenidos y comunicación directa con Xplico. Aplicando las políticas adecuadas conseguirá desplegar un sistema de clasificación de contenidos lo más robusto y eficaz posible. Los criterios de diseño e implementación serán vistos en capítulos posteriores.

- Procesador de contenidos: 'Plugins'

  Serán los extremos finales del sistema de clasificación, donde se llevará a cabo el procesado de los contenidos y la correspondiente asignación de prioridades a los mismos. Estos módulos software son totalmente independientes del Distribuidor, pero su diseño e implementación tienen que estar fuertemente ligadas con el Distribuidor, ya que este será quien marque las pautas, debido a que será él quien los invoque. De modo que, el Distribuidor deberá proporcionar un interfaz de programación de cara a los Plugins para que estos conozcan los parámetros de entrada con los que cuentan, y la salida que se espera de ellos. Respetando este interfaz el resto de implementación corre de parte del desarrollador, pudiendo utilizar las herramientas que cree necesarias. A estos módulos procesadores de contenidos de ahora en adelante los llamaremos *'Plugins'*.

  Además del diseño e implementación del PluginManager, para llevar a cabo la evaluación del mismo, se implementarán varios Plugins de ejemplo que nos permitirán chequear su correcto funcionamiento. Su diseño e implementación serán revisados en capítulos posteriores.

  Xplico y el PluginManager deberán situarse en la misma máquina, por lo tanto la comunicación de los mismos no requiere de la implementación de un sistema de comunicación, simplemente es necesario proporcionar a Xplico un interfaz de programación específico para que sepa como debe ser llamado el PluginManager y que parámetros son necesarios para su correcta ejecución. Sin embargo, la llamada a los Plugins por parte del PluginManager

como veremos más adelante puede ser local o remota, lo que implica la selección e implementación de un protocolo de comunicación basado en la tecnología de *WebServices* para este último caso.

Esta arquitectura basada en Plugins tiene como objetivo el desarrollo de un sistema modular que permita la inserción y eliminación de módulos software de forma sencilla, que dote al sistema de alta flexibilidad de desarrollo, provocando una mejor expansión. De modo que los desarrolladores podrán implementar Plugins que amplíen la funcionalidad del sistema de forma independiente al modulo de pre-clasificación principal a partir de los interfaces proporcionados.

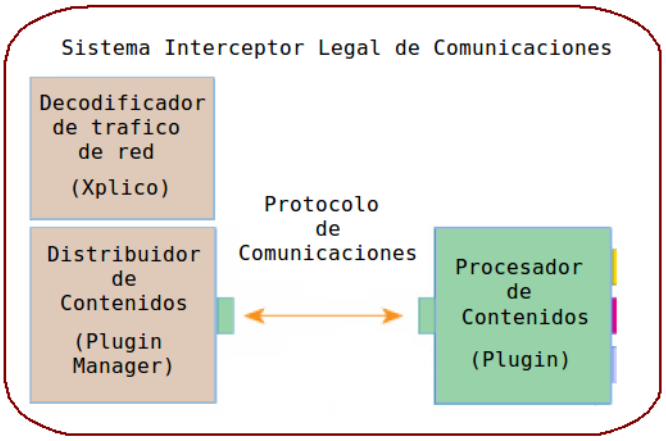


Figura 1.6: Arquitectura básica del Proyecto.

## 1.4. Metodología

Todo proyecto de desarrollo software debe girar entorno a una metodología de trabajo concreta, sistemática y predecible, que ayude al desarrollador a organizar el trabajo de la mejor forma posible y así optimizar el tiempo invertido en su implementación.

Este Proyecto se encuentra en colaboración con otros Proyectos finales de carrera (Implementación de Plugins) cuya realización depende directamente del presente proyecto. De modo que a lo largo de su diseño se han llevado a cabo diversas reuniones de equipo donde se han debatido las diferentes tecnologías y métodos para su implementación. Teniendo en cuenta la colaboración en el diseño del proyecto con un grupo de trabajo, la metodología que se ha seguido para su realización ha sido la presentada a continuación.

- **Estudio funcional y tecnológico del problema**

  Como punto inicial se llevará a cabo un análisis superficial del contexto de realización del proyecto, definiendo la funcionalidad básica del software a implementar y los requisitos generales del mismo, llegando a un acuerdo con el jefe de equipo (que en este caso corresponde al tutor del proyecto). Para ello se enmarcará la situación del proyecto, analizando por encima las tecnologías actuales de que se disponen y las opciones con las que se cuenta.

  Tras este análisis deberán haberse dejado planteados los objetivos y requisitos del proyecto.

- **Diseño**

  Llegados a este punto se llevará a cabo un estudio en profundidad de las diferentes formas en que se puede abordar el problema que enmarca el proyecto. Para ello se analizarán las diferentes tecnologías y herramientas actuales que mejor se ajustan para llegar a la solución más robusta y óptima posible. Se deberán dejar bien definidos los interfaces de entrada y salida de los que constará la aplicación, es decir, los requerimientos de entrada necesarios para poder ofrecer la funcionalidad descrita y los elementos de salida que generará tras su ejecución.

  Cuanto más detallado se encuentre el diseño más tiempo ahorraremos en la implementación, ya que el desarrollador tendrá unas pautas de trabajo ya predefinidas y sabrá de ante mano lo que está tratando. No obstante, el diseño inicial no es fijo, puede sufrir peque-

ñas modificaciones en el proceso de implementación, ya que según se va desarrollando los módulos surgen nuevas situaciones que en la planificación inicial no se habían contemplado.

Por tanto, el desarrollo final del diseño será el resultado de varias reuniones de equipo, dónde cada miembro expondrá diferentes puntos de vista a cerca de la mejor solución posible en cuanto a la resolución del problema.

- **Implementación**

  Una vez detallado el diseño se llevará a cabo la implementación del software del que se han debido dejar ya habladas las tecnologías, herramientas y pautas a seguir.

- **Evaluación de la solución**

  Una vez finalizada la implementación se llevará a cabo la comprobación de la validez de la solución elaborada, teniendo en cuenta que la solución se considerará válida si resuelve los problemas expuestos en el planteamiento del problema y satisface los objetivos definidos en la introducción. Dicha evaluación debe ser llevada a cabo tanto por el desarrollador del software, como por un miembro del grupo que se haya mantenido al margen de la implementación, el cual en este caso se trata del coordinador de proyecto. El desarrollador deberá pasará una batería de pruebas lo más específica posible para detectar el mayor número de errores y repararlos. El coordinador verificará que el software implementado es robusto y válido.

- **Documentación**

  Con el fin de eventuales correcciones, usabilidad, mantenimiento futuro y expansión del sistema, es de vital importancia documentar el desarrollo y gestión del Proyecto. Su construcción se basará en el lenguaje de modelado de sistemas de software *'UML'*, que incluirá diagramas, pruebas realizadas, manuales de usuario utilizados, etc.

Capítulo 2

# ESTADO DEL ARTE

## 2.1. Actividades ilícitas en Internet

Como todos bien sabemos Internet es un conjunto descentralizado de redes de comunicación interconectadas, donde la proliferación de nodos en todo el planeta ayuda a la difusión inmediata de los mensajes y permite el acceso a cualquier información introducida en la red. Esto dota a las personas de una herramienta potentísima que a simple vista puede aportar multitud de ventajas y efectos positivos, pero si nos detenemos a pensar, a estas ventajas se unen los malos hábitos que pueden tener lugar en el sistema si no se hace un adecuado uso de la información.

Debido a la popular aceptación de Internet como medio de comunicación, igualmente que ocurre al margen de la virtualidad de la red, existen 'lados oscuros' donde los usuarios aprovechan para llevar a cabo actos fuera de la ley. El correo electrónico, los sistemas de información basados en contenidos web, comercio electrónico, streaming de vídeo e incluso actualmente llamadas telefónicas sobre IP (VoIP) son medios de comunicación corrientes que se utilizan a diario. Por ejemplo algunas de las actividades ilícita que podemos encontrarnos hoy en día en Internet pueden ser las siguientes:

- Organización y comunicación de grupos delictivos a través de correo electrónico o voz IP.
- Envío masivo de spam, virus o fishing que atacan la privacidad de los usuarios de la red.
- Llamadas de voz VoIP, streaming de audio y vídeo que puedan transportar tráfico criminal.
- Sistemas de difusión P2P que puedan distribuir contenidos ilegales, tales como por ejemplo pornografía infantil.
- Conversaciones de chat (messenger, facebook, etc) que puedan ayudar a las autoridades a descifrar las claves de un caso.
- Páginas web que lleven a cabo acciones de fraude electrónico.
- Falsedad en documentos electrónicos.
- Uso ilegítimo de Passwords y entrada a sistemas informáticos no autorizados.

Estos son unos de los muchos ejemplos que pueden suceder en Internet y que mediante la interceptación legal sus responsables podrían ser llevados ante la Justicia.

## 2.2. Sistemas de interceptación legal de la información

Como consecuencia de los delitos mencionados en la anterior sección, los Cuerpos de Seguridad se han visto en la obligación de desarrollar sistemas de vigilancia virtual a través de los cuales garantizar la seguridad de sus ciudadanos. Estos sistemas de interceptación legal de la información son altamente secretos, únicamente conocidos por las Fuerzas y Cuerpos de Seguridad, ya que un pequeño conocimiento de las mismas podría ayudar a los infractores en la búsqueda de mecanismos para eludirlos. Es por ello que su infraestructura, funcionamiento y alcance es algo que permanece en el más absoluto secreto. Debido a su escasa información solamente podremos dar datos generales de los más famosos sistemas de interceptación de información actuales. Los más destacados son:

- **SITEL**

  Sistema Integrado de Interceptación de Comunicaciones Electrónicas (SITEL), fue desarrollado por la compañía Ericsson a lo largo del año 2000 para permitir al Gobierno Español interceptar llamadas telefónicas, siendo capaz de conocer las conversaciones mantenidas por los usuarios, la identidad del mismo, su operador telefónico y el punto geográfico exacto en el que se encuentra a la hora de la realización de las llamadas.

  La forma de actuar en la investigación de usuarios telefónicos por SITEL se muestra en las siguientes líneas:

  La interceptación de las comunicaciones llevadas a cabo son legales, ya que los cuerpos de seguridad para poder proceder necesitan de una orden judicial. Una vez obtenida dicha orden la policía comienza automáticamente la interceptación de las comunicaciones, siendo capaz de 'pinchar' varias líneas telefónicas al mismo tiempo.

  Las conversaciones telefónicas interceptadas se graban a través del software propietario de SITEL, que además registra todos los datos del usuario investigado. El agente de seguridad realiza las escuchas a partir del material almacenado en SITEL.

  Si en alguna conversación la policía encuentra la necesidad de presentarla ante el juez, hace una copia y la utiliza en los tribunales. Una vez que el tribunal verifica esta información, si decide que no es relevante se destruye, aunque no deja de ser una copia, ya que la conversación telefónica original sigue almacenada en la base de datos de SITEL.

- **Carnivore**

  Software utilizado por el FBI Norteamericano para capturar todo lo que un usuario transmite y recibe a través de su conexión a Internet. Tras la aceptación de una orden judicial el FBI tiene permiso para instalar el sistema de rastreo en los proveedores de acceso.

  Entre sus características conocidas caben destacar:

  1. Capacidad 'quirúrgica' de distinguir entre sujetos interceptados y no interceptados.
  2. Capacidad de distinguir entre datos interceptables de un sujeto y datos no interceptables, basándose en los poderes concedidos por la orden judicial de interceptación.
  3. Comparte información (al igual que ECHELON) con la industria, con el fin de desarrollar estándares adecuados a los requerimientos del sistema.
  4. Es posible que tenga relación con el sistema ECHELON o simplemente que sea una de sus partes inteligentes.

- **ECHELON**

  Es un sistema software de espionaje y análisis utilizado para la interceptación de comunicaciones electrónicas, cuyo desarrollo fue llevado acabo por cinco países (Estados Unidos, Canadá, Gran Bretaña, Australia, y Nueva Zelanda) integrantes del *'Convenio de Seguridad EE.UU.-Reino Unido (UK-USA)'*.

  Este proyecto se encuentra muy protegido, y a penas se sabe de su funcionamiento, pero como se ha podido leer en *wikipedia*, es sabido que tuvo su origen con el objeto de controlar las comunicaciones de la Unión Soviética y sus aliados, pero se sospecha que en la actualidad ECHELON ha sido actualizada para colaborar con la búsqueda de pistas sobre temas asociados al terrorismo, narcotráfico e inteligencia política y diplomática. Sus críticos afirman que el sistema está siendo utilizado también para el espionaje económico de otras naciones y la invasión de privacidad en gran escala.

  Varias fuentes afirman que estos estados han ubicado estaciones de interceptación electrónica y satélites para capturar gran parte de las comunicaciones establecidas por radio, satélite, microondas, móviles y fibra óptica. Las señales capturadas son luego procesadas por una serie de superordenadores, conocidas como diccionarios, las cuales han sido programadas para buscar patrones específicos en cada comunicación, ya sean direcciones, palabras, frases, o incluso voces específicas.

## 2.3. Problemática actual

La problemática que ocasiona la interceptación legal de comunicaciones es muy controvertida, ya que Internet es un espacio donde se intercambian una gran cantidad de contenidos de carácter lícito e ilícitos. Según la legislación actual Internet ya no es únicamente utilizada con fines académicos, empresariales o militares, sino que como ya hemos comentado a lo largo de este capitulo, dado a su excepcional proceso de expansión se ha abierto a toda la sociedad, dando lugar al surgimiento de un nuevo área de delincuencia que conlleva una forma diferente e indirecta de actuar difícil de controlar.

Para controlar la correcta utilización de internet desde hace tiempo se ha recurrido a la Regulación de la información, de la cual habrá quien esta a favor y quien esté en contra. Los partidarios de la regulación se apoyan en la tesis de que las redes de telecomunicaciones como Internet han generado un submundo en el que los delitos son difíciles de perseguir debido a la propia naturaleza del entorno. Sin embargo, frente a la corriente reguladora se reivindican los argumentos tales como el derecho a la intimidad y la libertad de expresión.

- **Regulación de las telecomunicaciones en Internet**

  La legislación referente a la Regulación de las Telecomunicaciones en Internet es muy extensa y puede hacerse muy pesado su análisis. Por lo tanto, con el objetivo de mostrar una idea superficial de la problemática actual generada por la regulación, se va a presentar a continuación un artículo comentado que refleja a la perfección la controversia ocasionada al utilizar sistemas de interceptación. El artículo está enfocado en la descarga de contenidos a través de redes P2P, uno de los marcos en los que se mueve este proyecto, pero no el único, ya que recordemos que la interceptación legal que se persigue en el presente proyecto también tiene como objetivo ayudar a los cuerpos de seguridad a resolver delitos del mundo real que utilizan Internet como medio de comunicación.
  El título del artículo y la dirección de publicación se muestran a continuación:

*'El software Híspalis de la Guardia Civil cada vez genera más controversia'*

http://www.teknoplof.com/2010/08/23/el-software-hispalis-de-la-guardia-civil-cada-vez-genera-mas-controversia/.

En él se nos cuenta la controversia de los 'Falsos Positivos' y el problema que genera al *'Grupo de delitos telemáticos de la Guardia Civil'* a la hora de clasificar los actos delictivos producidos por la descarga de contenido ilegal en las redes *'peer to peer'*.

Se expondrán algunas citas al artículo donde se comentarán las problemáticas relacionadas y mi opinión personal al respecto.

- *Híspalis es un programa informático que se encarga de rastrear las redes P2P en busca de contenido pedófilo con el objeto de poner a sus poseedores en manos de la justicia.*

  La ciega confianza por parte de la 'Guardia Civil' en estas herramientas provocan en algunos casos la injusta acusación de usuarios del sistema de intercambio de archivos *'peer-to-peer'* (eMule, bittorrent, etc). Según el articulo la política que ellos emplean a la hora de incriminar a alguien se muestra en las siguientes citas.

- *Híspalis tiene un rango de rastreo de 5 archivos pedófilos encontrados a un usuario único, esto es, hasta ese número se consideran tropiezos digitales y, a partir de él, intención directa.*

  Debemos estar de acuerdo en que la posesión de este tipo de material podría llevarnos ante un infractor, pero es de vital importancia estar seguros de que los contenidos relacionados han sido descargados voluntariamente antes de actuar, teniendo en cuenta que un alto número de descargas de contenido ilegal son accidentales.

  Cabe señalar que el criterio empleado para determinar si la posesión de material delictivo es accidental o intencionada resulta excesivamente rigurosa y que carece de sentido, puesto que un individuo puede tropezar fácilmente con tal número de archivos, sobre todo si se trata de un usuario inexperto.

  Deberían utilizar otro tipo de razonamiento, o mejor dicho, otro tipo de forma de clasificar la información obtenida por la aplicación.

  Lo mejor sería llegados a tal punto clasificar a estos usuarios como posibles sospechosos, pero nunca permitir que la herramienta decida de forma instantánea su culpabilidad. Esto quiere decir que el software utilizado no debería determinar si el usuario se ha descargado este material de forma accidental o intencionada, simplemente debería pre-clasificar al usuario como sospechoso, asignando un nivel de prioridad en función del número de ficheros delictivos descargados. Por tanto, considero fundamental que la decisión de acusación debería ser tomada por un humano, no por una máquina,

recalcando la idea de que el software únicamente debe clasificar la información y no tomar decisiones.

- *Híspalis funciona basándose en el identificador hash de los archivos compartidos en las redes de pares más utilizadas (eDonkey y FastTrack). La Guardia Civil española, en el momento de echar a andar a su Híspalis, disponía de 50.000 imágenes y vídeos de contenido pedófilo debidamente organizados y catalogados por grupos, que eran el resultado de años de investigación e incautación de material de este tipo a los infractores. Se tuvo en cuenta, de forma pionera, una serie de parámetros capaces de clasificar e identificar el amplio espectro de imágenes pedófilas que se mueven en la Red. Una vez ordenado el material, el mayor escollo que tuvo que salvar la Guardia Civil fue el de identificar uno por uno todos los ficheros mediante el indicador alfanumérico que proporcional la función hash.*

  Actualmente existen otros mecanismos relacionados con la generación de códigos para identificar de forma unívoca los contenidos de red que podrían devolver resultados bastante mejores que el proporcionado por la antigua función Hash (Ejemplo, Fuzzy Hashing).

- *El problema resulta de partir de la falsa premisa de que una dirección IP se corresponde con un usuario único. Para nada una IP puede ser considerada el DNI digital de un usuario, pues existen redes corporativas y empresariales en las que decenas de personas comparten una IP pública, ciber-cafés anónimos, redes Wifi públicas o sin proteger debidamente, servidores proxy, virus troyanos, etcétera. Además, y como segunda gran pega, la información que ofrece un meta-dato acerca de un usuario en una red de pares no es para nada indicativo de difusión dolosa y libidinosa, ya que la multitud de archivos pedófilos escondidos bajo una apariencia normal puede hacer que un internauta poco experimentado almacene (y, por tanto, comparta) material ilegal sin prácticamente conocimiento de ello.*

  La red P2P es la mejor herramienta que tienen los infractores para difundir todo tipo de material ilícito, ya que este tipo de descarga es utilizada por todo tipo de usuarios, desde los más avanzados hasta los más inexpertos. La utilización por usuarios inexpertos de estas herramientas de descarga de contenidos protagonizan los principales agujeros de material ilícito que los infractores utilizan para difundir de forma que no

se vean directamente implicados. Un usuario puede haberse descargado este tipo de material y estar compartiéndolo en la red sin darse cuenta.
Sabiendo esto no es muy coherente tomarse a la ligera la acusación/encarcelación de una persona por posesión de contenidos ilegales.

- *Otra de las quejas que enarbolan los detractores de Híspalis es la falta de rigurosidad judicial con la que se producen los escáneres. Aunque, para hacer honor a la verdad, este tema fue ya dirimido por el Tribunal Supremo en el año 2009 al dictar una sentencia en la que avalaba los rastreos que realizan las Fuerzas y Cuerpos de Seguridad del Estado en Internet para perseguir conductas delictivas, como el intercambio de ficheros con pornografía infantil.*

  En cuanto a este aspecto, todos estamos de acuerdo que se debe respetar la privacidad de las personas, y siempre se ha luchado por ello. Pero la gran pregunta es: ¿Hasta que punto se deben interceptar las comunicaciones para evitar actividades ilícitas en la red?. Estamos ante un tema delicado que genera controversia entre los usuarios, pues como en todo habrá multitud de opiniones y la solución final para esto no dejará nunca satisfechos a todos.

  Para analizar este aspecto, vamos a plantearnos la siguiente pregunta, ¿Qué posibilidades tiene un policía de investigar a un sospechoso en Internet sin invadir su privacidad?
  Por ejemplo, analicemos las acciones legales (con orden judicial) que un policía puede emplear físicamente (dejando al margen la virtualidad de internet) para comprobar si un individuo está realizando acciones delictivas. El policía puede vigilar al sospechoso para comprobar a que lugares va, con que gente se relaciona, si recibe o envía paquetes sospechosos, es decir, observar comportamientos atípicos de una persona que lleva una vida acorde con la ley.
  Pensemos ahora qué posibilidades tiene un policía de realizar esto mismo a través de internet... Observar a los lugares que va equivaldría a ver las webs que visita, con que gente se relaciona equivaldría a comprobar con quien intercambia mensajería instantánea o correos electrónicos, comprobar el envío/recepción de paquetes podría asemejarse a las descargas que realiza... Todo ello requiere un escaneo del tráfico cursado por el sospechoso.

Por tanto bajo mi punto de vista teniendo en cuenta la peligrosidad que supone la incorrecta utilización de internet, creo necesaria la permisión de los rastreos efectuados por los Cuerpos de Seguridad del Estado, siempre y cuando la toma de decisiones a cerca de las acusaciones realizadas se efectúen en base a criterios que entren dentro de la lógica y estén bien fundamentados.

Para el caso que nos atañe considero que las política empleadas por la Guardia Civil, no son nada efectivas y que deberían estudiar una profunda modificación.

- *¿Por qué provoca tanto rechazo Híspalis? Mi opinión personal es que es una herramienta con un fondo muy bueno pero mal utilizada. Es imposible dejar en manos de una máquina la adjudicación de la condición de pedófilo a una persona, condena que lo marcará socialmente de por vida, aunque su caso sea sobreseído. Y si las personas que están detrás de la máquina no hacen otra cosa que fiarse de ella, el resultado es un despropósito enorme. Teniendo en cuenta la cantidad de internautas novatos que pueblan la Red y los centenares de troyanos que se les cuelan a diario, una investigación más exhaustiva debería ser de obligatorio cumplimiento antes de entrar en una casa y confiscar los equipos informáticos*

Al escritor del artículo no le falta razón en cuanto a los dudosos criterios de análisis utilizados por la herramienta Híspalis. Estamos de acuerdo que lo que falla en el sistema no es la forma de trabajo de la herramienta empleada sino la forma de aplicar los resultados obtenidos por la misma. Como he comentado en puntos anteriores, de ningún modo se puede dejar la decisión de si un individuo es culpable de delito a una computadora, esta es una labor humana.

- **Conclusión obtenida**

Una vez analizada la problemática ocasionada por la regulación de la interceptación legal de información por las fuerzas de seguridad del estado, concluimos que el objetivo principal del Proyecto será la pre-clasificación de información de red, es decir, el etiquetaje de contenidos según la prioridad asociada tras su procesado. De tal modo que esta pre-clasificación ayude al personal de seguridad en la toma de decisiones a la hora de llevar a cabo una clasificación final y decidir si un individuo debe ser investigado en profundidad o simplemente se trata de una falsa alarma.

## 2.4. Web Services

Como se ha comentado anteriormente el dispositivo software a desarrollar presenta una arquitectura modular de plugins, los cuales se comunican con el nodo central de proceso mediante la tecnología de comunicación Web-Services. De modo que a continuación se va a dar una breve introducción a los servicios web y su funcionamiento.

- **Definición**

  Web Services (Servicios Web), son Sistemas Software que presentan una infraestructura independiente del lenguaje y plataforma, desarrollada para la comunicación entre aplicaciones de una forma desacoplada e interoperable sobre Internet.

  Analizando la descripción:

  - *Independiente del lenguaje y plataforma*

    Separación de la especificación y la implementación.

  - *Desacoplada*

    Posibilita la comunicación entre aplicaciones mediante el intercambio de mensajes de forma síncrona y asíncrona.

  - *Interoperable*

    Basado en estándares. Quiere decir que distintas aplicaciones, en lenguajes de programación diferentes, ejecutadas sobre cualquier plataforma, pueden utilizar los Servicios Web para intercambiar datos.

    La interoperabilidad se consigue mediante el uso de estándares abiertos. Se asientan sobre protocolos y estándares ya existentes y muy bien definidos (HTTP, XML, etc) que veremos más adelante cuando estudiemos su arquitectura.

    Uso de protocolos específicos extensibles, que no imponen restricciones sobre las aplicaciones a las que dan acceso ni sobre las tecnologías que las implementan (independencia del lenguaje y plataforma).

  - *Sobre Internet*

    Quiere decir que no existe control centralizado, es decir, se usan protocolos bien establecidos y consideraciones de seguridad.

En otras palabras, un Servicio Web es una aplicación software identificada por una URI, cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML, haciendo posible la interacción entre 'agentes' software (aplicaciones) utilizando mensajes XML intercambiados mediante protocolos estandarizados de Internet.

- **Conceptos clave**

  Antes de entrar a analizar los orígenes, características y arquitectura de los Servicios Web, vamos a recordar algunos conceptos clave para entender la motivación de su surgimiento y la gran importancia que están teniendo en las comunicación en Internet en la actualidad.

  - *Middleware*

    Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

    Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores. Abstraen al programador de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, a través de una API.

  - *Servicio*

    Un servicio es un procedimiento, un método o un objeto con una interfaz estable y pública que puede ser invocado por un cliente.

    Los Servicios Web amplían esa idea para permitir que esa invocación se realice a través de Internet empleando protocolos Web estándar ya existentes.

  - *Arquitectura orientada a Servicios (SOA)*

    Son arquitecturas que se aproximan al diseño de aplicaciones complejas basadas en la identificación de los servicios que ofrecerá, la definición de esos servicios y la organización de las interacción de esos servicios.

    Se basan en la idea de desarrollo del sistema a partir de interfaces. Estas interfaces permiten llevar a cabo un tratamiento automático para generar código de implementación que facilite la vida del desarrollador ahorrando tiempo y logrando desarrollar servicios y clientes eficientes.

    Los servicios ofrecen operaciones a los clientes a través de los interfaces que deben ser invocados en un orden determinado para lograr el objetivo deseado.

- **Historia**

  Desde los comienzos de las redes informáticas se ha tratado de solucionar el problema ocasionado en la comunicación de aplicaciones software localizadas en diferentes maquinas (Sistemas distribuidos). El problema viene motivado por los diferentes tipos de lenguajes de programación, plataformas hardware y sistemas operativos en que pueden correr las aplicaciones a comunicar.

  Anteriormente a la aparición de los Servicios Web, se habían realizado intentos de creación de estándares que estandarizaran la comunicación entre distintas plataformas a través de Internet, pero fracasaron o no tuvieron el suficiente éxito.

  Algunos de los más populares middlewares basados en programación orientada a objetos que surgieron para solucionar el problema fueron DCOM, CORBA y RMI. Estos pueden ser de gran utilidad en intranets donde se conoce perfectamente las plataformas utilizadas y se tiene control absoluto de los sistemas atravesados, pero como veremos a continuación existen claras desventajas en la utilización de los mismos en Internet:

  - *Dependencia del Vendedor*

    DCOM y CORBA dependen de la implementación del vendedor, DCOM-Microsoft y CORBA-ORB (a pesar que CORBA de múltiples vendedores pueden operar entre sí, hay ciertas limitaciones para aplicaciones de nivel superior en los cuales se necesita seguridad o administración de transacciones).

  - *Utilización de RPC*

    Estos middlewares hacen uso de RPC (Remote Procedure Call) para realizar la comunicación entre diferentes máquinas remotas. Esto, además de presentar ciertos problemas de seguridad, tiene la gran desventaja de que su implementación en un ambiente como Internet es casi imposible, debido a que tiene que atravesar multitud de sistemas intermedios hasta llegar a destino y muchos de ellos cuentan con firewalls que bloquean este tipo de mensajes, lo que hace prácticamente imposible a dos maquinas conectadas por Internet comunicarse.

  De modo que, los Servicios Web surgieron con la finalidad de poder llevar a cabo la comunicación entre diferentes plataformas, es decir, comunicar dos aplicaciones software de forma distribuida a través de Internet.

- **Arquitectura básica**

  OASIS y W3C son las organizaciones responsables de definir la arquitectura y estándares para los Servicios Web. Como hemos comentado en anteriores secciones, los Servicios Web presentan una arquitectura asentada sobre protocolos y estándares ya definidos, los cuales se presentan en la Figura 2.1.

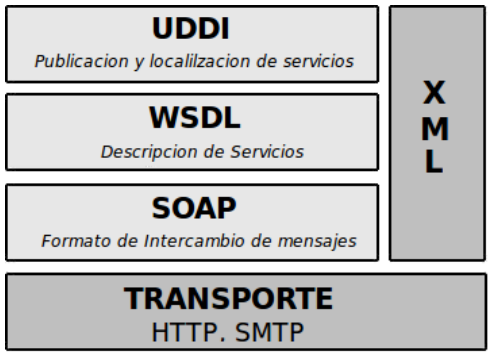


Figura 2.1: Web Services: Elementos necesarios para la definición de Servicios Web.

- XML (eXtensible Markup Language)

  Estándar desarrollado por el W3C que define un metalenguaje extensible de etiquetas, que se utilizará en los Servicios Web para especificar los lenguajes y protocolos necesarios. Permitirá la definición de lenguajes para:

  - *Describir servicios (WSDL)*
  - *Representar mensajes intercambiados (SOAP)*

- SOAP (Simple Object Access Protocol)

  Estándar desarrollado por el W3C que define mecanismos de interacción entre extremos. En su especificación se expone como organizar la información de forma estructurada y tipada usando XML para intercambiar información embebida en los mismos entre los extremos de la invocación.

  El Protocolo SOAP especifica:

  - *Un formato de mensajes común y extensible.*

    Describe como se organiza la información a intercambiar en formato XML.

  - *Conjunto de normas para implementar RPC mediante mensajes SOAP.*

- *Reglas a seguir por las entidades que procesen los mensajes SOAP.*

- *Descripción del modo en que se envían los mensajes SOAP sobre el protocolo de transporte (HTTP o SMTP).*

Sus características más importantes son las siguientes:

- *Define un intercambio de mensajes sin estado:*

  Posibilidad de soportar comunicaciones con estado añadiendo información adicional (IDs únicos) en la cabecera de los mensajes SOAP.

- *Define una comunicación en una sola dirección:*

  Los intercambios más complejos son gestionados por los extremos.
  Esquemas síncronos (RPC): mensaje petición más mensaje respuesta.
  Esquemas asíncronos: mensaje petición más llamada a función de callback.

- *No impone restricciones sobre la semántica de los mensajes intercambiados:*

  SOAP solo ofrece la infraestructura para transferir los mensajes, el significado de los mismos (etiquetas y datos) es interpretado por los extremos finales.

- *No se encarga de cuestiones de fiabilidad, integridad de los mensajes, transacciones, seguridad, etc:*

  La gestión de esos aspectos es responsabilidad de la infraestructura/aplicación que implementa el Servicio Web.

- *Bindings SOAP:*

  La especificación SOAP es independiente del protocolo de transporte usado para transferir los mensajes. SOAP sólo define un contenedor de 'mensajes' y la forma de encapsularlos en el protocolo de transporte que se haya decidido usar.
  La configuración del protocolo de transporte a utilizar se realiza modificando el atributo 'bingng' SOAP.

La estructura básica de los mensajes SOAP se presenta resumida en la Figura 2.2, la cual se ha descrito en la Tabla 2.2:

| Elemento | Descripción |
|---|---|
| <header> | Representa la cabecera del mensaje, siendo un elemento opcional. Contendrá información sobre el mensaje a usar por la infraestructura de Servicios Web: identificadores de transacciones, información de seguridad, etc. Puede encontrarse estructurado en bloques. |
| <body> | Representa el cuerpo del mensaje, siendo obligatorio su aparición. Contiene información específica a usar por las aplicaciones que usan o implementan el Servicio Web. Los extremos son los responsables de acordar el formato de la información intercambiada y de generar y/o procesar su contenido. |

Tabla 2.2: *Estructura de los mensajes SOAP.*

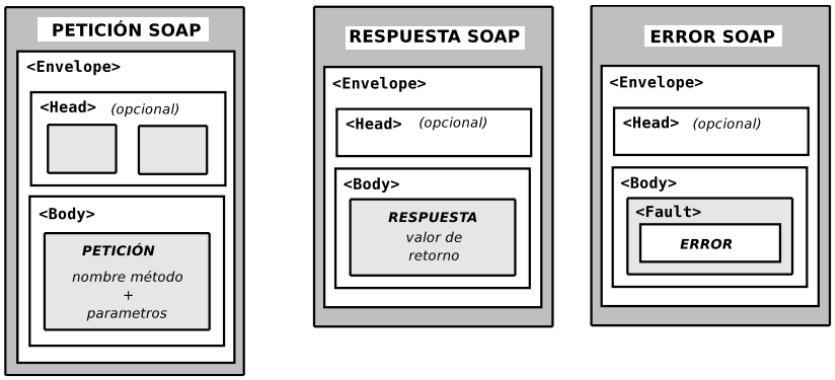


Figura 2.2: Web Services: Estructura de los mensajes SOAP.

Referencias http://ccia.ei.uvigo.es/docencia/SCS

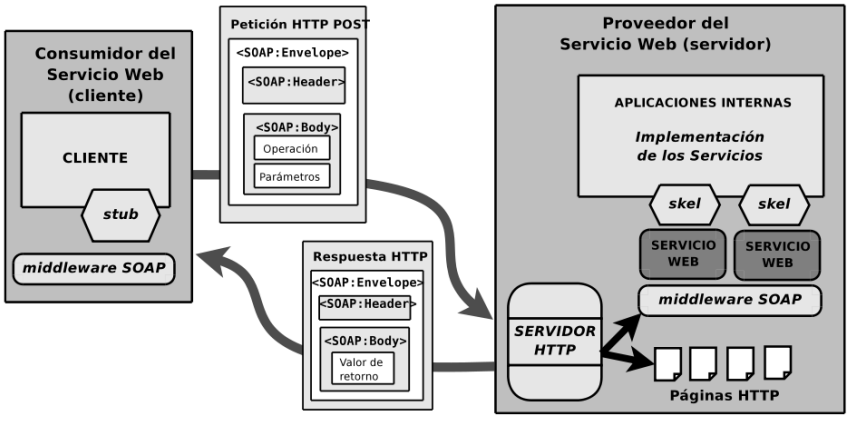


Figura 2.3: Web Services: Ejemplo de intercambio de mensajes SOAP.

Referencias http://ccia.ei.uvigo.es/docencia/SCS

- Web Service Description Language (WSDL)

  Lenguaje común utilizado para la descripción de Servicios Web y sus interfaces de forma estándar mediante documentos XML.

  Incluye toda la información necesaria para suplir la falta de un middleware común centralizado:

  - *Especifica cada operación disponible, con sus parámetros de entrada y de salida.*
  - *Puede usarse para generar los stubs/skeletons y las capas intermedias necesarias para escribir:*

    Clientes que invoquen los servicios web Servidores que los implementen.
  - *Especificar información sobre la localización de servicios (URIs).*

  El documento WSDL describe un servicio web como una colección de *ports* (puertos, extremos de la comunicación) capaces de intercambiar mensajes. La estructura básica de un documento WSDL se presenta resumida en la Figura 2.4, la cual se ha descrito en la Tabla 2.4:

| **Elemento** | **Descripción** |
|---|---|
| <types> | Define los tipos y estructuras de los datos intercambiados. Se utilizan los tipos definidos en la especificación de esquemas XML. |
| <message> | Define los mensajes (datos que van a ser intercambiados), indicando su nombre y su contenido. |
| <portType> | Define grupos de operaciones permitidas, y los mensajes intercambiados en el servicio. Para cada operación (elemento <operation>) se le asigna un nombre y se especifica el intercambio de mensajes. |
| <bindings> | Asocia a un grupo de operaciones (portType) una especificación de la codificación de mensajes y el el protocolo de transporte (atributo transport) que se va a utilizar en el intercambio de mensajes. |
| <service> | Especifica una agrupación lógica de puertos, indicando la URI (endpoints) donde se localizan los servicios correspondientes a las operaciones descritas a través de los elementos <portType>, con los que se intercambiarán las llamadas a través de mensajes SOAP. |

Tabla 2.4: *Elementos de una especificación WSDL.*

Con estos elementos definidos en la Tabla 2.4, no sabemos que hace un servicio pero si disponemos de la información necesaria para interactuar con él (funciones, mensajes de entrada/salida, protocolos, etc).

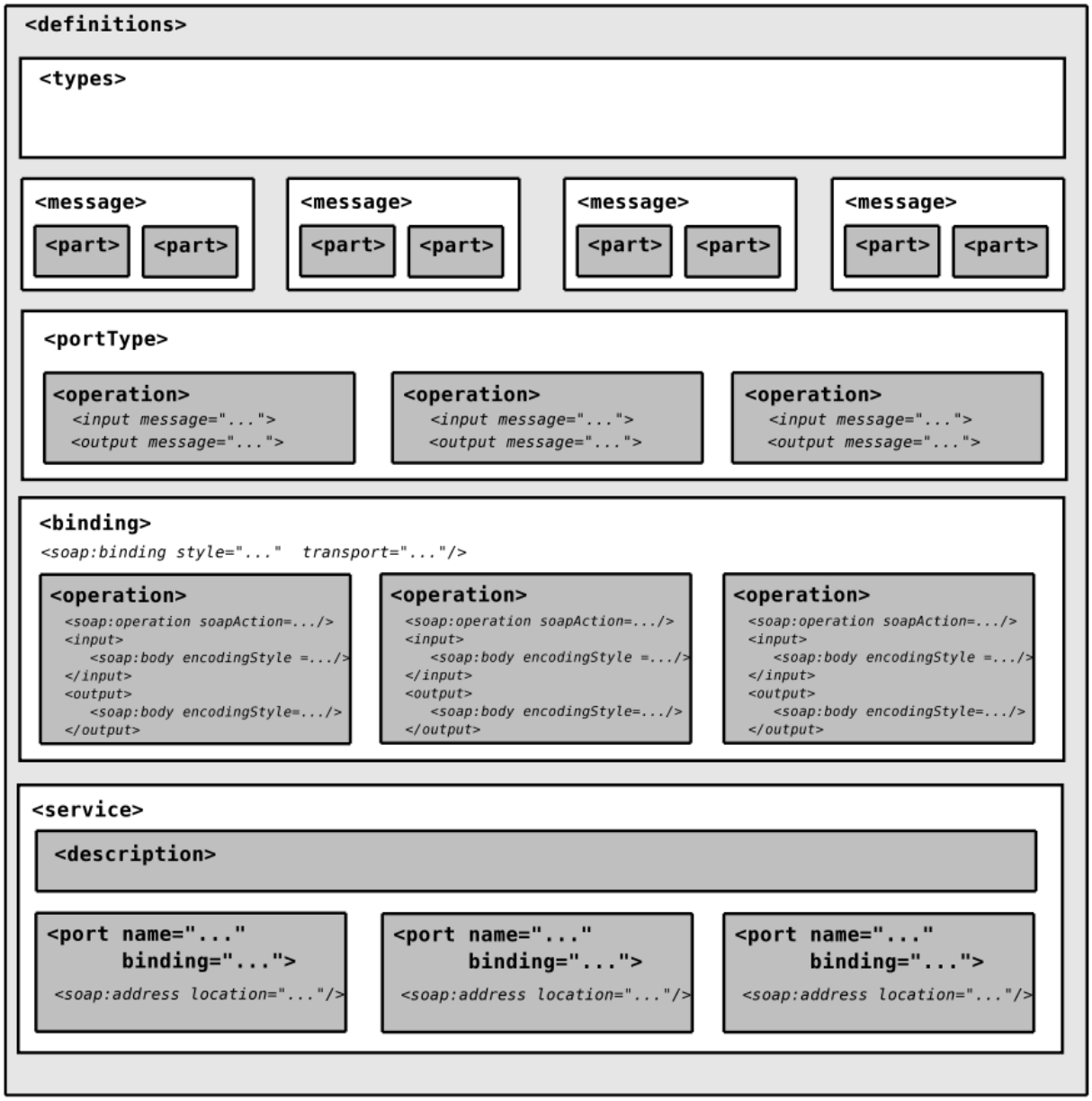


Figura 2.4: Web Services: estructura básica de un documento WSDL.

Referencias http://ccia.ei.uvigo.es/docencia/SCS

- Universal Description, Discovery and Integration (UDDI)

  Estándar desarrollado por la organización OASIS, que se encuentra relacionada con los Servicios Web como mecanismo de publicación y localización de servicios. Es un protocolo que sirve para interaccionar con un servidor (registro UDDI) que proporciona operaciones (vía mensajes SOAP) para registrar y descubrir Servicios Web.

  El protocolo UDDI especifica:

  - Cómo se publican y descubren los servicios.
  - Cómo trabajan los directorios de servicios web.

  Cada servicio se registra dando su nombre, una descripción del servicio (URL de su WSDL, descripción textual, etc).

  La utilización de UDDI se hace a través de APIs de programación basadas en SOAP.

  - El API se encuentra especificada con WSDL.
  - Alta (publicación) de Servicios Web por parte de servidores.
  - Localización (descubrimiento) de Servicios Web por parte de clientes.

  Finalidad:

  - Ofrecer soporte para encontrar información sobre Servicios Web y poder construir clientes.
  - Facilitar el enlazado dinámico, permitiendo consultar referencias y acceder a servicios de interés en tiempo de ejecución.

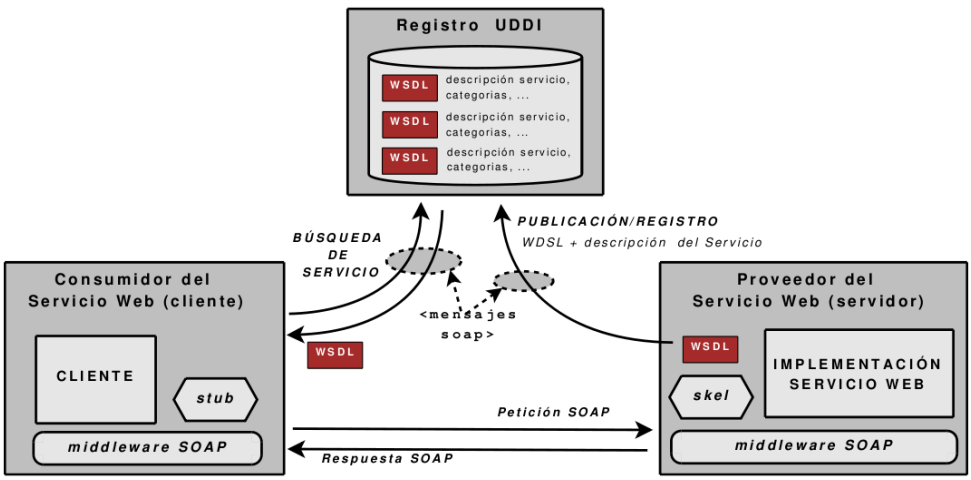


Figura 2.5: Web Services: Ejemplo interacción UDDI.

Referencias http://ccia.ei.uvigo.es/docencia/SCS

- Transporte

  Este intercambio de información se realizará en la mayoría de los casos sobre el protocolo HTTP. Se utiliza principalmente HTTP por ser un protocolo ampliamente distribuido y que se encuentra menos restringido por firewalls (generalmente se bloquean puertos como el FTP, pero el HTTP es altamente probable que no este bloqueado). A pesar de limitar mucho el uso de los Servicios Web al protocolo HTTP, éstos no fueron diseñados para ser utilizados sobre un único protocolo en particular, es decir, sus mensajes pueden viajar sobre cualquier otro protocolo de comunicación de Internet como SMTP, FTP, etc.

Esta arquitectura se basa en el modelo Cliente-Servidor, donde la comunicación entre aplicaciones se basará en el intercambio de información entre estas dos entidades:

- Aplicación Cliente

  Será aquella que efectúe las llamadas pertinentes al Servicio localizado en en una URI específica.

- Servicio Web

  Será la parte Servidora de la infraestructura encargada de escuchar peticiones de las Aplicaciones Clientes, procesar la información y devolver una respuesta si el carácter de la petición así lo requiere.

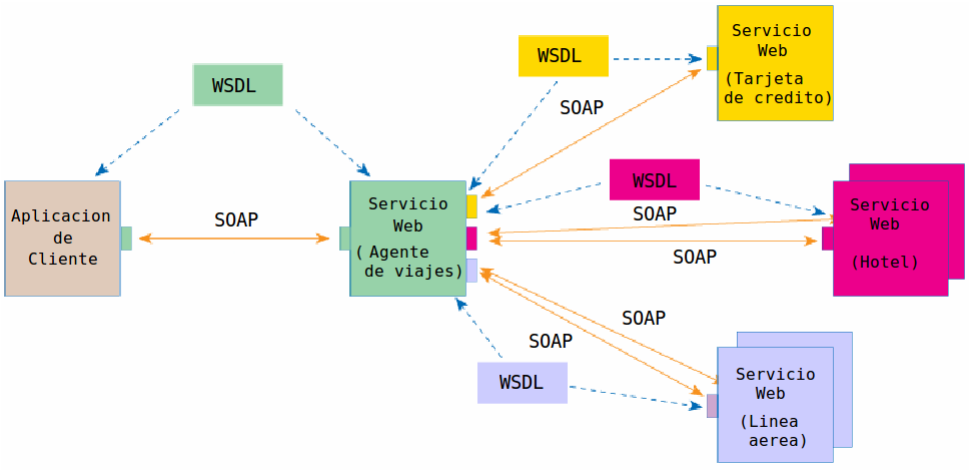


Figura 2.6: Web Services: Arquitectura de los Servicios Web.

- **Ventajas que ofrecen los Servicios Web**

  - Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
  - Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
  - Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

- **Inconvenientes**

  - Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
  - Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
  - Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.

- **Motivación de uso**

  La principal razón para usar servicios Web es que se basan en HTTP sobre TCP (Transmission Control Protocol) en el puerto 80. Dado que las organizaciones protegen sus redes mediante firewalls (que filtran y bloquean gran parte del tráfico de Internet), cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados.

  Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran ad-hoc y poco conocidas, tales como EDI (Electronic Data Interchange), RPC (Remote Procedure Call), u otras APIs.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más utilizada.

Capítulo 3

# DISEÑO DEL SISTEMA

## 3.1. Introducción

La motivación para la elaboración del presente proyecto, el atractivo que trata de convertirlo diferente del resto de sistemas de interceptación legal de comunicaciones viene fundamentado en la idea de elaborar un sistema capaz de pre-clasificar contenidos de red procesados de la mejor forma posible, siguiendo unas pautas que ayude al personal de seguridad a una clasificación final y posterior toma de decisiones a la hora de llevar a cabo una investigación más profunda contra individuos que utilizan Internet como mecanismo para llevar a cabo actividades ilícitas.

El artículo comentado en el Capítulo 2.3. representa un claro ejemplo de lo que tratamos de mejorar con el desarrollo de este Proyecto, tomar las experiencias y resultados de sistemas de interceptación como Híspalis para conseguir elaborar un sistema que no cometa sus mismos errores. Para ello el sistema se centrará en el campo de la pre-clasificación de contenidos de red de una manera óptima.

Además, la principal característica que lo hace diferente del resto es que actualmente no existe ningún sistema Europeo que procese contenidos de red, ya que como hemos visto únicamente existen sistemas que interceptan comunicaciones telefónicas (como SITEL).

Cabe destacar la disponibilidad en el PluginManager de un mecanismo de entrenamiento que permitirá al operador del sistema modificar las clasificaciones finales asignadas a los contenidos anteriormente pre-clasificados, consiguiendo con esto tener Plugins actualizados que poco a poco consigan pre-clasificar los contenidos de una forma más óptima.

En este capítulo se expondrán los criterios que se han seguido para diseñar el sistema de pre-clasificación y entrenamiento de contenidos llevado a cabo en el proyecto. Recordemos el objetivo y política de diseño principal visto en (1.3) para una mejor comprensión del capítulo:

- *Objetivo: 'aprovechar la información de tráfico de red proporcionada por la herramienta Xplico para pre-clasificar los contenidos capturados, generando un interfaz gráfico que muestre los resultados y permita a los usuarios su análisis y clasificación final'.*
- *Política de diseño: 'modulo software perfectamente integrable con la herramienta Xplico'.*

## 3.2. Arquitectura de la herramienta Xplico

La herramienta de análisis forense de tráfico de red Xplico cuenta con una arquitectura compuesta por cuatro macro componentes los cuales pueden verse en la Figura 3.1, y se comenta a continuación.

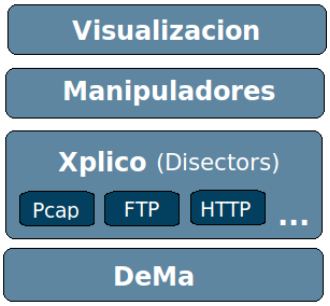


Figura 3.1: Arquitectura de la herramienta Xplico.

- *DeMa (Decoding Manager)*

  Componente encargado de la decodificación de los datos de red capturados, llevada a cabo a través de las siguientes funciones:

  1. Organización de los datos de entrada.
  2. Establecimiento de la configuración e historial de archivos para la decodificación.
  3. Lanzamiento de decodificador y manipuladores.
  4. Control del decodificador y lanzamiento de manipuladores.

- *Xplico*

  Decodificador compuesto de diversos módulos llamados *'Disectors'* encargados de la extracción de información específica de protocolos del tráfico capturado y gestionados por el *'DeMa'*. En la actualidad Xplico cuenta con una gran cantidad de *Disectors* correspondientes a los protocolos más comunes (Figura 1.1).

- *Manipuladores*

  Conjunto de herramientas y aplicaciones para la manipulación de los datos decodificados: transcodificación, correlaciones y agregadores.

- *Visualización*

  Sistema de visualización para ver los datos extraídos. No es mas que un interfaz web desarrollado en CakePHP dónde se muestran de forma ordenada los datos extraídos de la decodificación y posterior extracción de información de los protocolos disponibles. Muestra la información de forma menos detallada que los comunes analizadores de tráfico como *'whireshark'*, *'tcpdump'*, etc; mostrando la información más relevante que permita de una forma más amigable, y rápida el análisis del tráfico que ha sido capturado.

La relación entre los diferentes componentes comentados se puede apreciar en la Figura 3.2.

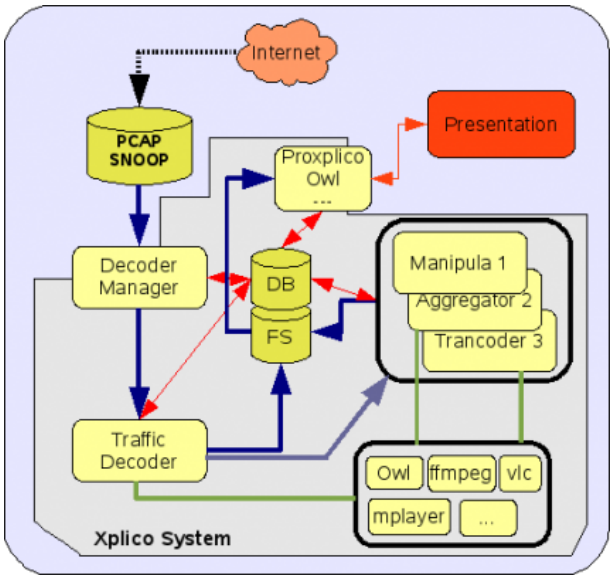


Figura 3.2: Xplico: relación entre sus diferentes macro componentes.
Referencias http://wiki.xplico.org/doku.php?id=architecture

No entraremos más en detalle en cuanto a Xplico se refiere puesto que como ya veremos en la conclusión descrita al final del documento la integración con esta herramienta no pudo llevarse a cabo debido al tiempo que esto requiere, ya que para ello se necesita una reunión y puesta de acuerdo con el desarrollador de Xplico para encontrar la mejor forma de integración. Dicha integración quedará reflejada como un trabajo futuro que ya comentaremos más adelante.

## 3.3. Arquitectura del PluginManager

En esta sección se presenta la estructura básica del sistema de clasificación de contenidos que se ha desarrollado, con la finalidad de mostrar una idea básica y tener una representación del modelo, que ayude a la comprensión de éste capítulo. La Figura 3.3, se verá complementada con la descripción detallada del diseño de cada componente en las siguientes secciones del capítulo.

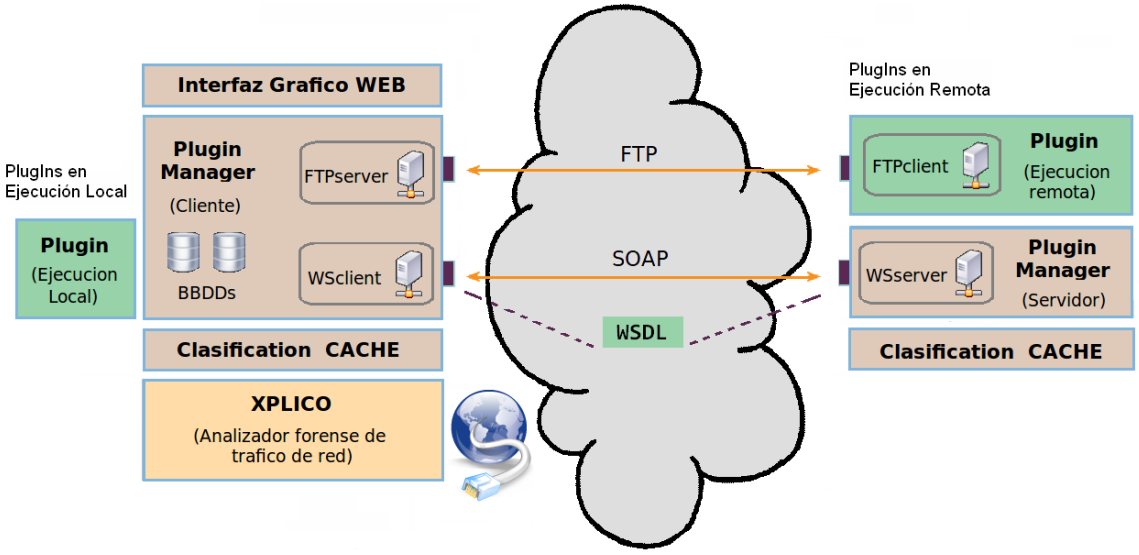


Figura 3.3: Arquitectura del Sistema de clasificación de contenidos de red.

Como se puede ver en la Figura 3.3, el sistema presenta tres módulos principales:

- *Xplico:* herramienta forense de análisis de tráfico de red (ver Capítulo 1.2).
- *PluginManager:* módulo software encargado de la distribución de contenidos de red proporcionados por la herramienta Xplico entre los diferentes plugins disponibles, capacitados para su pre-clasificación.
- *Plugins:* módulos software que contienen la lógica necesaria para procesar y etiquetar contenidos de red.

La finalidad del presente proyecto es la elaboración del módulo PluginManager, aunque para comprobar su correcto funcionamiento y validez se han diseñado varios Plugins de ejemplo. En las siguientes secciones del capítulo se detallarán los criterios escogidos para su implementación y comunicación con el resto del sistema.

## 3.4. Diseño de la Comunicación Intermodular

Existen diversas formas de comunicar dos módulos software, entre ellas cabe destaca la comunicación mediante:

- Uso de memoria compartida.
- Sockets.
- Web Services.
- Transferencia de ficheros.
- Llamadas a procesos locales.
- Otros...

Abstrayéndonos de los diferentes sistemas de comunicación que se han citado, podemos clasificarlos todos ellos en dos grupos:

- *Ejecución Local*

  Las llamadas al modulo software en cuestión se llevan a cabo en el mismo equipo que se sitúa la herramienta solicitante.

- *Ejecución Remota*

  Las llamadas al modulo software en cuestión se llevarán a cabo a través de la red, estando ubicados la herramienta solicitante y el modulo llamado en diferentes máquinas.

A continuación explicaremos los criterios de selección que se han tenido en cuenta para la selección de las tecnologías y métodos de desarrollo para la comunicación entre los diferentes módulos del sistema expuestos en la sección anterior.

- **Comunicación Xplico-PluginManager**

  Como ya veremos más adelante en la Sección 3.5, el modulo PluginManager proporciona un interfaz de programación específico que ofrece a Xplico la funcionalidad necesaria para el procesado y pre-clasificación automática de contenidos de red. Xplico utilizará este interfaz para obtener el resultado de clasificación de los contenidos. El modo en que Xplico llamará a estas funciones especificadas en el interfaz proporcionado por el PluginManager queda de la mano del desarrollador de Xplico por lo que no entraremos en más detalle.

- **Comunicación PluginManager-Plugins**

  - **Tipos de Plugins atendiendo a la forma de llamada**

    Independientemente de todas las tecnologías disponibles que existen para la comunicación intermodular, se ha decidido que el PluginManager debe ser capaz de interactuar con Plugins tanto en el ámbito local como remoto. Los motivos que han llevado a esta decisión se exponen a continuación.

    - **Plugins de ejecución Local**

      Representa la forma de llamada más básica, sencilla y eficaz que se puede realizar. La llamada se lleva a cabo de forma directa ya que ambos módulos se encuentran situados en la misma máquina. Este tipo de ejecución es la indicada para aquellos Plugins cuyo tamaño, funcionalidad y requerimientos de recursos y procesamiento son reducidos, aquellos que permitan ser ejecutados de una forma rápida y no pongan en peligro la fluidez de ejecución del PluginManager.
      En la gran mayoría de los casos esta llamada suele requerir la previa configuración de parámetros del sistema, como puede ser la configuración del *'classpath'* especificando la ubicación de las librerías necesarias para la correcta ejecución del mismo. De modo que la llamada se verá encapsulada a través de la ejecución previa de un script, que configurará los parámetros necesarios y finalmente llevará a cabo la ejecución del Plugin.

    - **Plugins de ejecución Remota**

      Existen diversos motivos que han llevado a decidir la necesidad de desarrollar un mecanismo de llamada remota a Plugins.
      El principal motivo que encontramos es: la sobrecarga que provocaría la ejecución local de un Plugin cuya demanda de recursos de la máquina es elevada, lo cual comprometería la ejecución fluida de Xplico y del PluginManager, relentizando el sistema.
      Por otro lado, pensemos que el sistema de interceptación legal de comunicaciones que utilice el PluginManager como modulo de pre-clasificación de contenidos no va a ser único, sino que es muy probable que se emplee más de un dispositivo que

utilice el módulo en diferentes puntos de la red. De modo que, si existen varios plugins que son utilizados por todos los PluginManager estaríamos replicando plugins innecesariamente generando información duplicada. En estos casos la mejor alternativa es situar estos plugins de uso común en otra máquina y que todos los PluginManager que estén siendo utilizados por los sistemas de interceptación legal los llamen mediante llamadas remotas a través de la red. Así conseguiremos centralizar el procesamiento en un único nodo y la información de las bases de datos, evitando la duplicación de información.

- **Tecnología empleada para cada tipo de plugin**

  La llamada a plugins de ejecución local no requiere de ningún sistema especial ya que se llevará a cabo a través de una simple llamada al binario que implementa su funcionalidad, obteniendo el resultado de pre-clasificación por salida estándar.

  En cuanto a la llamada a plugins remotos, se han estudiado las tecnologías actuales que más se usan y se ha decidido la utilización de Servicios Web (WS, Web Services). Las razones de esta elección serán explicadas en las siguientes lineas.

  - *Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.*

    Útil para conseguir un sistema distribuido y poder ubicar los Plugins en máquinas independientes al 'PluginManager'. De esta forma, los cambios a lo largo del tiempo en un modulo (PluginManager o Plugin) no deben afectar al otro. Esto aporta al sistema gran flexibilidad a la hora de su mantenimiento y actualización, debido a que resulta más cómodo, rápido y eficaz mantener una única máquina (donde se aloja el Plugin) que un montón de máquinas (donde se alojan los plugins locales comunes al PluginManager).

  - *Fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.*

    Aprovecharemos la utilización de XML para embeber la información de los contenidos de red y los resultados proporcionados por los plugins tras su análisis. La utilización de un protocolo XML facilitará el crecimiento del sistema, debido

a que la modificación de los campos de los paquetes de peticiones y respuestas intercambiados entre PluginManager y Plugin será tan fácil como incluir unas simples etiquetas al documento XML transportado a través de SOAP.

- *Se apoya en HTTP, lo cual permite atravesar los sistemas de seguridad (firewall) sin necesidad de cambiar las reglas de filtrado.*

  Esta es una de las principales razones por las cuales la utilización de servicios web para la comunicación remota se está poniendo tan de moda. Las organizaciones protegen sus redes mediante firewalls, que filtran y bloquean gran parte del tráfico de internet, cerrando casi todos sus puertos excepto el 80, que es precisamente el que usan los navegadores. Los servicios web utilizan este puerto por la simple razón de evitar ser bloqueados y llegar a todo el mundo.

- *Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinadas fácilmente para proveer servicios integrados.*

  De este modo podremos añadir servicios adicionales de forma rápida y sencilla. Esta característica es especialmente interesante para el proyecto INDECT, el cual se encuentra formado por 17 socios de diferentes países Europeos.

Sin embargo su aplicación también conlleva una serie de desventajas que tendremos que tener en cuenta y que en ocasiones nos llevarán a decidir si merece la pena situar un Plugin en local o en una máquina remota.

Todos aquellos Plugins que demanden unos requisitos computacionales elevados como podrían ser el de tratamiento de imágenes o descompresión de ficheros, es conveniente que sean desplegados en máquinas independientes al PluginManager, ya que estando en local consumirían gran cantidad de recursos y harían a la aplicación principal lenta.

Por el contrario existirán Plugins cuya complejidad y coste computacional requerido sea ínfimo y no merezca la pena desplegar todo el sistema necesario para poder gestionar un *Web Service*, ya que como veremos más adelante necesitaríamos disponer de un Servidor Apache en una máquina independiente que probablemente el coste de mantenimiento y flujos de datos de red intercambiados sean, en comparación con el resultado obtenido, muy elevados.

**Conclusión:** llegamos a la conclusión de diseñar un sistema híbrido, es decir, que de soporte tanto a Plugins en ejecución local como a Plugins en ejecución remota a través de *Web Services*.

- **Despliegue de Plugins Remotos (Servicios Web)**

  Como se ha explicado en la sección anterior, la comunicación con los plugins remotos se realizará mediante *Web Services*. De este modo cada Plugin será un Servicio Web, que tendrá que ser desplegado en un Servidor para poder ser accesible a través de la red por el PluginManager.

  El Servidor puede ser cualquiera, siempre y cuando soporte la funcionalidad para el despliegue de Servicios Web. Para el desarrollo de la comunicación mediante *Web Services* se ha escogido el Proyecto Axis de Apache, el cual ofrece toda la funcionalidad necesaria para la correcto despliegue de los mismos. Se ha escogido Axis porque es el proyecto de desarrollo de *Web Services* más popular y que la inmensa mayoría de desarrolladores utilizan en la actualidad, y por lo tanto de la que más documentación hay disponible.

  Para la implementación del sistema y la realización de pruebas, se ha escogido el Servidor Apache Tomcat junto a un modulo Axis2 que aporta la funcionalidad básica para el tratamiento con servicios web. La elección de Tomcat se debe a que es un Servidor Web reducido, fácil de configurar y de poco tamaño, que añadiéndole simplemente un pequeño modulo de Axis (que explicaremos en un Anexo más adelante) proporciona la funcionalidad que necesitamos para desplegar todo el sistema de comunicación.

- **Apache Axis Proyect**

  Apache Axis es un middleware que aporta una simple abstracción al programador para la implementación de Servicios Web basados en SOAP.

  Apache Axis and Axis2 fueron desarrollados como subproyectos del Proyecto 'Apache Web Service'. Como resultado de su exitoso desarrollo se ha terminado convirtiendo en un Proyecto independiente de alto nivel, al que se le ha llamado 'Apache Axis Proyect'. La página oficial donde podemos consultar toda su documentación es:

  http://axis.apache.org/.

- **Axis1.x**

  Apache Axis ha sido el motor de desarrollo de servicios web de código abierto más exitoso de los últimos tres años. Surgió como una continuación del proyecto 'Apache SOAP', utilizado para la interconexión entre extremos.

  Axis1.x permite el desarrollo de una infraestructura de servicios web completa programada en Java, y en las últimas versiones de la misma se comenzó a implementar para C++.

  Se encuentra basada en una arquitectura de 32 bits.

  Debido a los enormes cambios que tuvieron lugar en el marco de los Servicios Web, dieron lugar a la aparición de la segunda generación, que veremos a continuación.

- **Axis2**

  Axis2 es la nueva generación de código abierto para la implementación de Servicios Web, el 'core engine' más popular y ampliamente utilizado por los desarrolladores. Ofrece más funcionalidades, apoyo y actualizaciones. También permite desarrollar aplicaciones para arquitecturas de 64 bits, a diferencia de la generación anterior. A grandes rasgos los cambios más importantes respecto a la primera generación son:

  - No asume la interacción entre los extremos basada en petición-respuesta, sino que admite una amplia gama de interacciones de mensajería. Permite a los usuarios modelar la interacción con los extremos utilizando patrones de mensajería ilimitados. Ofrece soporte sólo para la ida (peticiones), utilizando como patrones de respuesta las API proporcionadas por el Cliente.
  - Permite comunicación entre extremos síncrona y asíncrona. El comportamiento de interacción puede configurarse fácilmente a través del API cliente.
  - Presenta un nuevo modelo de procesado de documentos XML con el que se encapsula y lee la información embebida en los mismos y que viajará por la red a través de SOAP. Para ello cuenta con la colaboración del 'Axiom Proyect' que aporta la funcionalidad necesaria para su tratamiento.
  - Se basa en un modelo de implementación relacionado con el modelo J2EE. Define un formato de servicio de archivos nuevo, simple y fácil de utilizar. Permite desplegar servicios simplemente copiando los archivos en el directorio del repositorio. Esto favorece al despliegue aislado de servicios proporcionando

una facilidad de uso cada vez mayor.

- Capacitado para desarrollar aplicaciones para arquitecturas de 64 bits.
- Desarrollado tanto para Java como para C/C++:
  http://axis.apache.org/axis2/java/core/index.html.
  http://axis.apache.org/axis2/c/core/index.html.

- **Acceso a los Contenidos de red proporcionados por Xplico**

  Xplico será quien gestione los contenidos de red capturados y el encargado de desgranar la información de los mismos y pasársela al PluginManager. Una vez que el PluginManager ha sido llamado dispondrá de la información necesaria para localizar y descargar el contenido seleccionado para analizar. La cuestión es la siguiente: *¿Cómo accede a estos contenidos los Plugins cuando son llamados para procesarlos?*. Existen varias formas de hacer que los Plugins consigan los contenidos que se les indica para procesar y pre-clasificar:

  1. *Pasar el contenido directamente en la petición.*
     Presenta la ventaja de que el Plugin se quita trabajo, puesto que el contenido le llega directamente y no tiene que descargarlo de ningún sitio. Sin embargo se están consumiendo recursos de red y haciendo que las peticiones tarden más en llegar, dando lugar en muchas ocasiones a peticiones duplicadas.

  2. *Pasar datos necesarios para que el Plugin sea capaz de descargar el contenido.*
     Presenta la ventaja de que las peticiones son muy ligeras y se envían con rapidez. La contrapartida es que el Plugin deberá ser capaz de descargar el contenido a partir de la información proporcionada y además el PluginManager tendrá que desplegar algún sistema para servir los contenidos a los Plugins y que estos puedan descargarlos.

  Analizando las ventajas y desventajas de estos dos métodos de intercambio de contenidos expuesto, se ha decidido desechar la opción de enviar el contenido directamente en la petición debido a que a parte de las desventajas ya nombradas, puede que los plugins no necesiten descargar el contenido a analizar porque disponen de un mecanismo de caché que consultan para comprobar si el contenido ha sido analizado antes y en tal caso no necesitarían descargarlo. Si esto ocurre estaríamos transmitiendo un

contenido por la red innecesariamente, pues en destino no se ha utilizado. No tendría mucha importancia si los contenidos a procesar tienen un tamaño pequeño, pero si por el contrario los contenidos son muy pesados (tales como imágenes o vídeos), estaríamos haciendo un uso ineficiente de los recursos, relentizando nuestra aplicación innecesariamente.

De modo que, una vez concluido el análisis decidimos optar por pasar al Plugin sólo la información necesaria para que este sea capaz de descargarlo si lo necesita. Esta opción trae consigo la necesidad de desplegar un mecanismo que sirva los contenidos. Para ello utilizaremos un simple sistema cliente-servidor FTP debido a su simplicidad, facilidad de uso y a que es uno de los más eficientes en cuanto a la descarga de ficheros pesados. Otra opción podría haber sido descargar los contenidos a través de HTTP configurando un servidor Apache, pero por eficiencia hemos decidido utilizado FTP.

## 3.5. Diseño de Interfaces de comunicación intermodular

Recordemos que el PluginManager es un módulo intermedio en el proceso de clasificación de contenidos de red, estando en uno de los extremos Xplico quien le proporciona la información necesaria de localización y caracterización de los contenidos, y en el otro extremo los Plugins, los cuales a partir de la información aportada por Xplico y otra adicional generada por el PluginManager deben ser capaces de procesar los contenidos y generar una pre-clasificación. De este modo el PluginManager ofrecerá dos interfaces de comunicación, uno a Xplico, y otro a los Plugins.

A continuación vamos a explorar la funcionalidad ofrecida por el PluginManager, la información que es necesaria para que el procesamiento de contenidos sea lo más óptimo posible y las diversas posibilidades de implementación que darán lugar a los interfaces de comunicación.

- **Funcionalidad ofrecida por el PluginManager a Xplico**

  El PluginManager Básicamente ofrece dos modos de operación que Xplico puede utilizar a través del interfaz que detallaremos más adelante.

  - **Análisis de un contenido**

    A través de esta función Xplico será capaz de pre-clasificar un contenido si dispone de los plugins adecuados para ello. Esta pre-clasificación se basa en el etiquetaje del contenido con un valor de prioridad que oscila entre los valores -1 y 5.

    - Prioridad -1, indica que el contenido no pudo ser procesado debido a un Error.
    - Prioridad 0, indica que el contenido no pudo ser pre-clasificado.
    - Prioridad [1-5], indican de forma gradual la importancia del contenido (1- irrelevante, 5- muy importante).

    Independientemente de la prioridad asignada en la pre-clasificación todo contenido debe ser procesado por el operador del sistema para establecer una clasificación final.

  - **Entrenamiento de los Plugins**

    Xplico podrá utilizarlo para modificar explícitamente la prioridad asociada a un contenido de red procesado. Esta funcionalidad permitirá la expansión del sistema haciendo que los Plugins cada vez sean capaces de procesar más contenidos y de forma más rápida. Permite modificar las prioridades que han sido asignadas a un contenido por los Plugins, sustituyéndolas por la indicada por el personal de seguridad que está utilizando Xplico.

- **Información necesaria para el procesamiento de contenidos de red**

  Atendiendo al marco de trabajo en el que se sitúa el PluginManager, en el ámbito de la interceptación legal de comunicaciones, que será llevada a cabo por los cuerpos de seguridad de un país, podemos determinar una serie de datos que los Plugins de procesamiento necesitarán a la hora de procesar y clasificar los contenidos. Estos campos son:

  - *Case category:* representará la categoría del caso al que pertenece el contenido. Necesario para ubicar un contenido en una categoría determinada como por ejemplo podría ser la de tráfico de drogas, armas, pedofilia, etc. Xplico deberá proporcionar explícitamente esta información.

  - *Case identification:* representará la identificación del caso. Identificador que representa e identifica unívocamente un contenido dentro del caso al que se encuentra asociado. La forma que tendrá dicho identificador queda de la mano de Xplico, el cual deberá proporcionar esta información explícitamente.

  - *Related contents:* contenidos relacionados con el caso, que podrían ayudar al procesado y posterior clasificación final por el operador. Por ejemplo en un caso de análisis de imágenes, se detecta que dicha imagen se ha pre-clasificado con un valor muy importante debido al gran parecido que presenta con una imagen localizada en la base de datos de contenidos ilícitos de la policía. En este caso en el campo de *'Related Contents'* irá la dirección URL donde se encuentra localizada la imagen a la que se parece. De este modo el operador encargado de procesar el resultado podrá comparar ambas imágenes y llevar a cabo una clasificación final adecuada.

  - *Content attributes:* información adicional que podría ayudar al procesado y posterior clasificación. Campo añadido con vistas de futuro, el cual podrá ser empleado si es necesario transmitir campos adicionales a los Plugins para completar la pre-clasificación.

  Atendiendo a las necesidades del PluginManager y Plugins, para completar con éxito el procesado de contenidos son necesarios una serie de datos que Xplico debe proporcionar en las llamadas a las funciones ofrecidas. Estos campos de información básicos son:

  - *Content Path:* necesario para localizar el contenido localmente y poder construir posteriormente la URI que se le pasará al Plugin para que pueda descargarlo por FTP.

- *Content Name:* nombre del contenido.
- *Content Type:* tipo de contenido a analizar en base al formato MIME-type.
- *Content Timestamp:* marca de tiempo que representa cuándo el contenido fue almacenado en el sistema.

Atendiendo a la forma en que el PluginManager recibe peticiones y como las gestiona para devolver una respuesta, éste puede ofrecer a Xplico sus funcionalidades de dos modos, de forma Síncrona o Asíncrona.

- Modo Síncrono:

  En este tipo de llamada Xplico lanzará una petición contra el PluginManager y se mantendrá a la espera hasta que el PluginManager despliegue toda su lógica y obtenga un resultado que devolver como respuesta a Xplico. Con este método Xplico permanecerá inutilizado durante el proceso de un Contenido, no pudiendo lanzar nuevas peticiones sobre cualquier funcionalidad ofrecida por el PluginManager hasta que no se haya recibido una respuesta de la anterior petición.

  La implementación de este modo como mecanismo de procesado de contenidos aporta la ventaja de que el PluginManager se ahorra la gestión de información de contexto de Xplico, ya que al procesar las peticiones de forma secuencial y síncrona, las respuestas de estas llegarán de tal forma que Xplico siempre conocerá a que contenido pertenece la respuesta recibida. Por contrapartida obtenemos un sistema muy lento e ineficiente, ya que en la red hay multitud de contenidos cuyo procesado es muy pesado y lento.

- Modo Asíncrono:

  Esta modalidad de implementar las llamadas permite a Xplico utilizar todas las funcionalidades ofrecidas por el PluginManager de forma continua, una tras otra, sin necesidad de esperar la recepción de la respuesta de una para emitir la siguiente.

  Las ventajas aparentes de este mecanismo son la posibilidad de que Xplico pueda continuar desempeñando tareas mientras los contenidos se están procesando. Este sistema requiere de un mecanismo de gestión de Contexto, puesto que al procesar contenidos en paralelo unos se clasificarán antes que otros, dándose la situación (muy probable) de que las respuestas recibidas de pre-clasificación lleguen en distinto orden a como sus peticiones fueron emitidas. De modo que el PluginManager debe ofrecerle

la posibilidad a Xplico de pasarle un Contexto que devolverá íntegro en las respuestas y que permitirá a Xplico conocer a que contenido pertenece esa respuesta recibida.

Como podemos ver la utilización del Modo Síncrono es mucho más sencilla y rápida de implementar, pero ofrece un rendimiento que no nos interesa ya que deseamos desarrollar un PluginManager que facilite la vida a Xplico lo máximo posible y que su utilización no lastre el rendimiento de Xplico. Por lo tanto se ha optado por utilizar un mecanismo Asíncrono en el procesado de los contenidos, que obliga a la incorporación de un campo de información en todas las peticiones, el cual es opaco para el PluginManager:

- *Contexto:* contendrá toda la información que Xplico considere necesaria para la identificación del contenido procesado y su personal tratamiento.

Por último, para aquellas peticiones que soliciten la funcionalidad de entrenamiento, Xplico necesitará pasar al PluginManager un valor que determine la nueva prioridad a asociar al contenido especificado. Este último campo de información queda caracterizado como:

- *Content Priority:* nueva prioridad asociada al contenido especificado. valor numérico que esté dentro del rango [-1,5]

- **Interfaces ofrecidas por el PluginManager**

  Después de haber analizado la información necesaria, modos de procesamiento, y demás aspectos de gestión de contenidos, podemos concluir esta sección resumiendo como quedan las interfaces de comunicación intermodular ofrecidos por el PluginManager a los diferentes módulos del sistema (Xplico y Plugins).

  - **Interfaz ofrecida por el PluginManager a Xplico**

    Este interfaz de comunicaciones será el que Xplico utilizará a modo de API para disponer de las funciones necesarias a las que llamará para ser capaz de utilizar toda la funcionalidad ofrecida por el PluginManager. Como ya se ha analizado, estas funciones requieren de una serie de campos de entrada y otros de respuesta. A continuación se presentarán una serie de tablas que mostrarán dicho interfaz, como resultado del análisis de diseño:

    - Funcionalidad de Análisis de contenidos: Tabla 3.2
    - Funcionalidad de Entrenamiento de contenidos: Tabla 3.4
    - Valores retornados por el Plugin tras su llamada: Tabla 3.6

| Campo de entrada | Descripción |
|---|---|
| Case category | Categoría del caso al que pertenece el contenido. |
| Case identification | Identificación del caso. |
| Content Path | Referencia a un contenido señalando la localización del mismo. |
| Content Name | Nombre del contenido a procesar. |
| Content Type | Tipo del contenido a procesar, expresado como un formato MIME-type (ej.: image/jpeg). |
| Content Timestamp | Marca de tiempo que representa cuándo fue almacenado el contenido en el sistema. |
| Related contents | Contenidos relacionados con el caso que podrían ayudar al procesado y posterior clasificación. |
| Content Attributes | Información adicional que podría ayudar al procesado. |

Tabla 3.2: *Interfaz PluginManager-Xplico: Funcionalidad de Análisis*

| Campo de entrada | Descripción |
|---|---|
| Case category | Categoría del caso al que pertenece el contenido. |
| Case identification | Identificación del caso. |
| Content Path | Referencia a un contenido señalando la localización del mismo. |
| Content Name | Nombre del contenido a procesar. |
| Content Type | Tipo (MIME-type) del contenido a procesar. |
| Content Description | Descripción asociada por el operador. |
| Content Priority | nueva prioridad ([1,5]) asociada al contenido. |
| Content Timestamp | Marca de tiempo que representa cuándo fue almacenado el contenido en el sistema. |
| Related contents | Contenidos relacionados con el caso que podrían ayudar al procesado y posterior clasificación. |
| Content Attributes | Información adicional que podría ayudar al procesado. |

Tabla 3.4: *Interfaz PluginManager-Xplico: Funcionalidad de Entrenamiento.*

Independientemente de la función a la que Xplico llame, la salida será la misma:

| **Campo de retorno** | **Descripción** |
|---|---|
| Content URI | Referencia a un contenido señalando la localización del mismo. |
| Content Hash | Secuencia hash que representa de forma unívoca a un contenido. Aunque como explicaremos más adelante para identificar de forma unívoca a un contenido utilizaremos además de este campo otros. |
| Content Name | Nombre del contenido a procesar. |
| Content Length | Longitud del contenido. |
| Process Timestamp | Marca de tiempo que indica cuando fue procesado el contenido. |
| Content Priority | Prioridad asociada por el plugin ([-1,5]) al contenido procesado. |
| Process Message | Descripción generada por el Plugin tras el procesar el contenido. |
| Related Contents | Contenidos relacionados con el caso. |
| Xplico CONTEXT | Información de contexto de Xplico. |

Tabla 3.6: *Interfaz PluginManager-Xplico: Respuesta generada.*

- **Interfaz ofrecida por el PluginManager a los Desarrolladores de Plugins**

  El PluginManager deberá proporcionar un interfaz que contendrá las funcionalidades anteriormente descritas, que deberán implementar los desarrolladores de Plugins para ofrecer dicha funcionalidad. Estas funciones tendrán unos campos de entrada que necesitarán los plugins para procesar los contenidos, y unos campos de salida que retornarán los plugins una vez procesado el contenido y que serán de utilidad para la clasificación del mismo. A continuación se presentarán una serie de tablas que mostrarán dicho interfaz, como resultado del análisis de diseño:

  - Funcionalidad de Análisis de contenidos: Tabla 3.8
  - Funcionalidad de Entrenamiento de contenidos: Tabla 3.10
  - Valores retornados por el Plugin tras su llamada: Tabla 3.6

| Campo de entrada | Descripción |
|---|---|
| Case category | Categoría del caso al que pertenece el contenido. |
| Case identification | Identificación del caso. |
| Content URL | Referencia a un contenido señalando la localización del mismo. |
| Content Name | Nombre del contenido a procesar. |
| Content Type | Tipo del contenido a procesar, expresado como un formato MIME-type. |
| Content Timestamp | Marca de tiempo que representa cuándo fue almacenado el contenido en el sistema. |
| Related contents | Contenidos relacionados con el caso que podrían ayudar al procesado y posterior clasificación. |
| Content Attributes | Información adicional que podría ayudar al procesado.. |

Tabla 3.8: *Interfaz PluginManager-Plugins: Funcionalidad de Análisis*

| Campo de entrada | Descripción |
|---|---|
| Case category | Categoría del caso al que pertenece el contenido. |
| Case identification | Identificación del caso. |
| Content Path | Referencia a un contenido señalando la localización del mismo. |
| Content Name | Nombre del contenido a procesar. |
| Content Type | Tipo (MIME-type) del contenido a procesar. |
| Content Description | Descripción asociada por el operador. |
| Content Priority | nueva prioridad ([0,5]) asociada al contenido. |
| Content Timestamp | Marca de tiempo que indica cuando fue entrenado el contenido. |
| Related contents | Contenidos relacionados con el caso. |
| Content Attributes | Información adicional que podría ayudar al entrenamiento. |

Tabla 3.10: *Interfaz PluginManager-Plugins: Funcionalidad de Entrenamiento.*

Independientemente de la función a la que el PluginManager llame, la salida será la misma, y además contendrán los mismos campos que la salida que será posteriormente generada desde el PluginManager para Xplico, (Tabla 3.6).

- **Resumen**

  Una vez explicadas las decisiones de diseño que han dado como resultado los interfaces de comunicación expuestos, en la Figura 3.4 podemos ver un ejemplo ilustrado que ayudará a una mejor comprensión. Se verá como el PluginManager se encuentra separado en dos, Cliente/Servidor (desde el punto de vista de los *'Web Services'*):

  - La parte Cliente interactuará con Xplico, proporcionando un interfaz a modo de API que utilizará para acceder a las funcionalidades de Análisis y Entrenamiento.
  - La parte Servidora actuará como una capa de adaptación entre las llamadas realizadas por el PluginManager a nivel de transporte y el Plugin solicitado. Se ha decidido que las llamadas a los Plugins pasen primero por esta capa con el objetivo de simplificarle la vida a los desarrolladores de Plugins, ya que esta capa gestionará los contextos, y desgranará las peticiones sirviendo los datos de entrada a los plugins de una forma más sencilla. Esta parte Servidor ofrecerá un interfaz de comunicaciones a los desarrolladores de plugins, que deberán implementar.

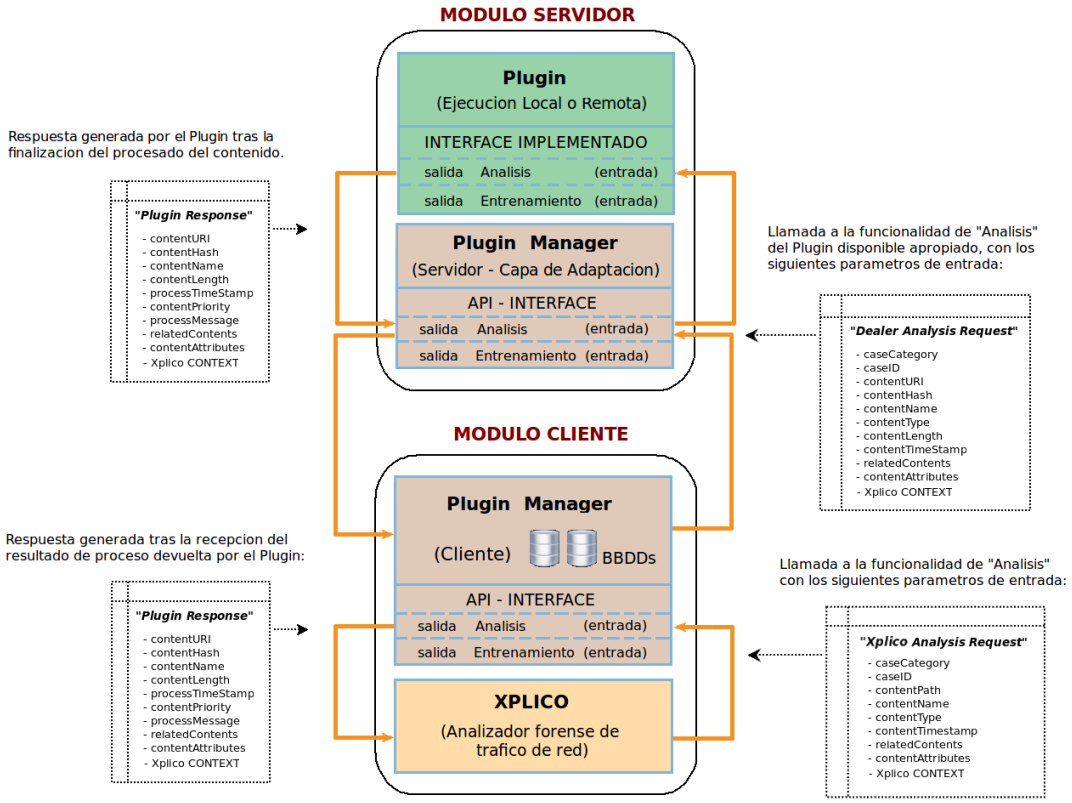


Figura 3.4: Interfaces de comunicación: consecución de llamadas e información.

## 3.6. Diseño del Modelo de información del PluginManager

El PluginManager contará con una base de datos que utilizará para almacenar información relativa al Registro de Plugins, Listas de operación, y Cache de contenidos.

De entre todos los sistemas de gestión existentes se ha decidido que la implementación de esta base de datos se lleve a cabo a través del sistema de gestión de bases de datos SQLite, debido a que el sistema de almacenamiento que utiliza la herramienta Xplico se encuentra diseñado con esta tecnología. De modo que debido a que el PluginManager está orientado a ser utilizado por la herramienta Xplico, se ha querido ser concordante con las tecnologías utilizadas y se ha ajustado a ellas para que la aplicación sea lo más integrable posible.

A continuación veremos cada una de las tablas que contiene la base de datos diseñada y que información se presenta en cada una de ellas.

- **Registro de Plugins**

  Una de las tablas presentes en la base de datos del PluginManager es la que contiene el Registro de los Plugins. En esta aparecerán todos los Plugins que han sido dados de alta en el sistema y que están disponibles para procesar contenidos. Los Plugins serán dados de alta por el operario del sistema y para ello deberá conocer ciertos datos del mismo como por ejemplo su nombre, ubicación (URI), etc. Los campos de cada registro pueden verse resumidos en la Tabla 3.12 situada al final de la sección.

- **Listas de operación**

  El PluginManager cuenta con una lógica de ejecución de Plugins tal que dependiendo del tipo de contenido a analizar y la disponibilidad de los plugins registrados la secuencia de llamadas a plugins será una u otra.

  El PluginManager utilizará esta tabla para conocer si dispone de Plugins para procesar un tipo de contenido determinado, y si a ese tipo de contenidos se ha configurado una lista de ejecución de plugins.

  A través de estas listas podremos configurar que ciertos tipos de contenidos sean analizados por más de un plugin. Por ejemplo, a la llegada de un correo electrónico podremos determinar que primero se analice si se trata de Spam, si este resultado da positivo descartaríamos

el contenido, si por el contrario da negativo pasaríamos a analizarlo a través de otro plugin que por ejemplo procese el texto que contiene, y así progresivamente hasta que el correo electrónico quede procesado en su totalidad.

Estas listas además aportan la posibilidad de ejecutar plugins de forma secundaria si el configurado por defecto no se encuentra operativo. Por ejemplo, para análisis de imágenes, imaginemos que el PluginManager tiene registrados dos plugins, uno que analiza imágenes de tipo *'.png'* (*'image/png'*) y otro que es capaz de analizar todo tipo de imágenes (*'image/*'*). Podemos configurar en las listas de operación que todas las imágenes de tipo *'image/png'* sean primero procesadas por el primer plugin descrito, y en caso de no haber podido obtener una pre-clasificación pasar al segundo plugin que es capaz de analizar cualquier tipo de imágenes. De este modo si uno de los dos plugins configurados se encuentra fuera de servicio, siempre habrá un segundo plugin para procesarlo.

Las listas de ejecución girarán en torno al tipo de contenido a analizar: *'image'*, *'message'*, *'text'*, etc. Igual que el registro de plugins estas secuencias de ejecución de plugins establecidas en esta tabla serán configuradas por el operario del sistema.

La forma que presentan los registros de listas de ejecución en la base de datos se muestra en la Tabla 3.14.

- **Cache de contenidos**

  El PluginManager cuenta con una caché de contenidos que utilizará para evitar llevar a cabo el análisis de peticiones de procesamiento a Plugins sobre contenidos que ya han sido procesados anteriormente. Esta caché se encuentra localizada tanto en el lado del Cliente como en el lado del Servidor (en cuanto a Plugins Remotos, es decir, Web Services).

  La identificación unívoca de un contenido se ha determinado que será llevada a cabo a través de la consecución de tres campos: 'content hash', 'content name' y 'content length'. Se ha decidido que utilizando el conjunto de estos tres campos no existe duda alguna a la hora de identificar un contenido. Podemos ver la forma de almacenar un registro de contenido en la caché en la Tabla 3.16.

  Esta tabla será únicamente modificable por el PluginManager, que será el único encargado de su actualización. El operario del sistema únicamente podrá limpiarla, pero en ninguno de los casos podrá añadir entradas a la misma.

| Campo | Descripción | Ejemplo |
|---|---|---|
| Plugin ID | Identificador del Plugin. | 1 |
| Plugin Name | Nombre asociado al Plugin. | Spamassassin |
| Plugin Type | Tipo de contenidos que analiza. | message/rfc822 |
| Plugin URI | Ubicación del Plugin. | http://localhost:8080/axis2/services /spamassassin. |
| Plugin Description | Descripción del Plugin. | Plugin that analyzes email messages. |

Tabla 3.12: *Base de datos PluginManager: Registro de Plugins.*

| Campo | Descripción | Ejemplo |
|---|---|---|
| RuleID | Identificador de regla. | 1 |
| Plugin ID | Identificación del Plugin. | 1 (Spamassassin) |
| Plugin Order | Orden de ejecución del Plugin para el Analysis Type en cuestión. | 1 |
| Content Type | Tipo de contenido a analizar. | message/rfc822 |

Tabla 3.14: *Base de datos PluginManager: Listas de operación.*

| Campo | Descripción | Ejemplo |
|---|---|---|
| Content ID | Identificador del contenido. | 1 |
| Content Hash | Secuencia Hash que representa el contenido. | 44d11b0f1daf55edfac723ea95a5c8d4 |
| Content Length | Tamaño del contenido. | 15539 |
| Content Priority | Prioridad asociada al contenido. | 2 |
| Process Message | Descripción asociada al procesado del contenido. | stored by training method. |

Tabla 3.16: *Base de datos PluginManager: Caché de contenidos.*

## 3.7. Diseño del PluginManager

- **Decisión del Lenguaje de programación empleado**

  Fieles a la política en la que se viene insistiendo a lo largo del proyecto, se ha estudia el diseño del PluginManager en base a conseguir un modulo perfectamente integrable con la herramienta Xplico. Sabiendo que Xplico se encuentra desarrollada utilizando como lenguaje de programación 'C', pensamos que la opción más lógica para la implementación del PluginManager sería utilizar 'C'. Sin embargo a medida que se han ido analizando las diversas tecnologías a utilizar (axis) para poder desarrollar el modulo, se ha llegado a la conclusión que una alternativa perfectamente integrable con Xplico y que facilitaría la implementación del PluginManager es 'C++'. Finalmente escogimos 'C/C++' como lenguaje de programación base para el PluginManager.

- **Diseño de la parte Cliente**

  El modulo software que representa la parte cliente del PluginManager se encuentra constituida por dos capas, una capa superior encargada de la lógica interna de procesamiento y una capa inferior de transporte que lo comunica con los distintos plugins del sistema. La Figura 3.5 ilustra la pila básica de capas de la que se compone y cuyo diseño detallaremos a continuación.

  - **Capa Lógica**

    Esta capa será la que contendrá la funcionalidad básica ofrecida por el interfaz de comunicación con el que interactuará Xplico. Tal y como muestra la Figura 3.5, la capa lógica será quien interactúe de forma directa con la base de datos del sistema, llevando a cabo las siguientes acciones:

    - Consulta de disponibilidad de Plugins para el análisis de contenidos.
    - Consulta y actualización de caché.
    - Gestión de las listas de procesado de Plugins.
    - Interacción con la capa de transporte para el lanzamiento de peticiones.

  - **Capa de Transporte**

    Esta capa se encuentra dividida en otras tres sub-capas. La principal de ellas es la 'capa de adaptación', la cual se comunica de forma directa con la capa lógica la que le pasa la información necesaria para establecer comunicación con el plugin seleccionado. Las

otras dos capas se orientan al modo en el que se llama al plugin indicado por la capa lógica, de ejecución local o remota. Dependiendo del modo de ejecución se llevarán a cabo unas acciones u otras, como por ejemplo la configuración del modulo Axis para la comunicación remota o la llamada a los scripts y módulos necesarios para las ejecuciones locales.

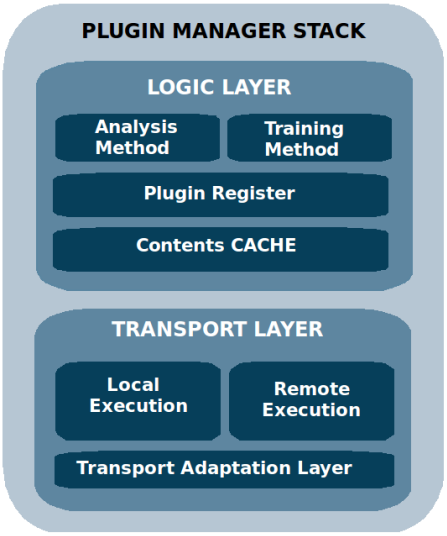


Figura 3.5: PluginManager stack: capas de las que se compone el modulo software.

- **Diseño de la parte Servidora (Capa de adaptación al Plugin)**

  El PluginManager cuenta con un modulo software situado en el lado del Plugin, que representa la parte servidora y que actúa como una capa de adaptación intermedia entre el PluginManager y el Plugin. Éste módulo recibe las peticiones de procesado y adapta la información recibida para posteriormente hacer la llamada al plugin de tal forma que simplifique lo máximo posible su labor.

  Contará con una caché de contenidos que consultará para determinar si la solicitud de procesamiento del contenido especificado ya ha sido procesado con anterioridad. De este modo se evita la llamada al Plugin innecesariamente. Si el resultado de la consulta a la caché resultó positiva, cogerá el valor de respuesta almacenado y devolverá lo devolverá al módulo Cliente. Si por el contrario no se encontraron coincidencias, procesará la petición recibida construyendo las estructuras necesarias para realizar una llamada al Plugin y que este lo procese.

## 3.8. Diseño de Plugins

El objetivo de este proyecto no es el diseño de Plugins de procesado de contenidos, pero si lo es la especificación detallada de cómo un desarrollador de Plugins debe actuar para implementar un Plugin de forma correcta y que pueda ser dado de alta en el PluginManager y ser llamado adecuadamente.

Como ya se ha visto en secciones anteriores, los plugins pueden clasificarse según la localización o forma de ser llamados, habiendo plugins de ejecución local y plugins de ejecución remota. Con el objetivo de simplificar al máximo la labor de los desarrolladores, se ha diseñado un sistema que abstraiga totalmente a éstos de la forma de ejecución de los mismos. Es decir, se pretende que al desarrollador de plugins no le afecte la forma en la que el plugin se va a comunicar con el PluginManager.

Para conseguir esto se ha definido un interfaz de comunicación en la Sección 3.5 de cara al Plugin, que deberá ser implementado por este. De este modo, el desarrollador únicamente tendrá que preocuparse de implementar las funciones descritas en dicho interfaz y seguir una serie de pasos expuestos en el ANEXO C (presente al final del documento) para conseguir desplegar el plugin adecuadamente.

En cuanto al lenguaje de programación utilizado para su desarrollo, dependiendo de la forma de ejecución del plugin podrá ser diferente. Todos ellos podrán ser implementados utilizando Java. Para aquellos plugins que vayan a ser ejecutados remotamente inicialmente obligamos a los desarrolladores de Plugins a que su implementación se realice utilizando Java, debido a que la capa de adaptación en el lado del Servidor del PluginManager (la cual podemos apreciar en la Figura 3.4) se encuentra implementada en Java y se espera que los Plugins también lo estén. Sin embargo, si el Plugin es de ejecución local, éstos podrán ser desarrollados en cualquier lenguaje, ya que las llamadas desde el PluginManager al comando que implemente el Plugin abstraerían a éste del lenguaje utilizado.

De modo que los objetivos a largo plazo serían diseñar una capa de adaptación en otro lenguaje de programación como podría ser C/C++ para dar la posibilidad a los desarrolladores de plugins a diseñar sus Plugins eligiendo el lenguaje que más les convenga, aunque dicha funcionalidad se ha dejado como trabajo futuro.

## 3.9. Diseño del PluginManager GUI

Para probar el correcto funcionamiento del sistema se ha diseñado un Interfaz Gráfico Web que actuará en lugar de la herramienta Xplico. Lo ideal sería integrar el módulo PluginManager con Xplico, pero eso supondría que tendríamos que ponernos de acuerdo con el desarrollador de Xplico y la falta de tiempo lo impide. Por ello este interfaz web actuará a modo de Xplico y podrá verse que el modulo implementado es perfectamente usable y realiza a la perfección todas las funcionalidades que se han ido describiendo en este capítulo.

Este interfaz web se ha decidido implementar utilizando CakePHP que es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Los motivos que han llevado a su elección han sido principalmente porque Xplico cuenta también con un interfaz gráfico web y éste se encuentra desarrollado con CakePHP. De modo que siguiendo la idea de la futura integración con Xplico, cakePHP ha sido el lenguaje elegido para diseñar el interfaz.

Capítulo 4

# IMPLEMENTACIÓN DE LA SOLUCIÓN DISEÑADA

Este capítulo tratará de explicar los aspectos más importantes de la implementación llevada a cabo del sistema diseñado en el capítulo anterior.

## 4.1. Contenidos del Proyecto implementado

En esta sección se puede ver la estructura de ficheros del proyecto desarrollado (Tabla 4.2).

| **Directorio** | **Descripción** |
|---|---|
| bin/ | Contiene ejecutables del proyecto. |
| conf/ | Ficheros de configuración. |
| db/ | Contiene la base de datos del PluginManager. |
| doc/ | Documentación del proyecto. |
| gui/ | Código fuente del interfaz gráfico cakephp. |
| lib/ | Contiene aquellas librerías de software externo utilizadas. |
| plugins/ | Contiene Plugins de ejemplo implementados para pruebas. |
| src/ | Contiene el código fuente de todo el Proyecto. |
| src/db/ | Clases necesarias para interactuar con bases de datos de tipo SQLite3. Interfaz programado que utiliza las librerías '*SQLite3*' para el tratamiento con bases de datos. |
| src/pm.client/ | Código fuente del PluginManager: representa la lógica principal de procesado del módulo software y la parte cliente de llamadas a plugins. |
| src/pm.server.plugins/ | Código fuente para desarrolladores de plugins. Cuenta con clases que actúan de servidor de PluginManager para adaptar la información recibida. |
| src/tomcat.embedded | Código fuente java que embebe un servidor Tomcat7. |
| test/ | Recursos empleados para la realización de pruebas de clasificación. |

Tabla 4.2: *Base de datos PluginManager: Implementación del Registro de Plugins.*

A continuación se presenta la estructura a modo de árbol, dónde se puede ver jerárquicamente todas las clases de cada módulo implementado.

```
.
|-- bin
|   |-- pm.client
|   |   |-- pluginManagerClient
|   |   `-- pluginManagerClient.sh
|   `-- pm.server.plugins
|       |-- plugins.tomcat.embedded.jar
|       `-- tomcat.embed.startup.sh
|-- conf
|   `-- (...)
|-- db
|   `-- pluginManager_database
|-- doc
|   `-- Pluginmanager.pdf
|-- gui
|   `-- (...)
|-- install.sh
|-- lib
|   `-- (...)
|-- plugins
|   |-- image.Analyzer
|   |   |-- bin
|   |   |   `-- (...)
|   |   |-- lib
|   |   |   `-- (...)
|   |   |-- resources
|   |   |   `-- services.xml
|   |   |-- run.as.local.sh
|   |   `-- src
```

```
|   |       `-- org
|   |           `-- xplico
|   |               |-- plugin
|   |               |   |-- XplicoPlugin_interface.java
|   |               |   `-- XplicoPlugin_main.java
|   |               `-- pluginmanager
|   |                   |-- adaptionLayer
|   |                   |   |-- local
|   |                   |   |   `-- LocalPluginAdaptionLayer.java
|   |                   |   `-- remote
|   |                   |       |-- PgCallbackHandler.java
|   |                   |       |-- PgMessageReceiverInOut.java
|   |                   |       `-- PgSkeleton.java
|   |                   `-- common
|   |                       |-- AnalysisRequest.java
|   |                       |-- PluginResponse.java
|   |                       `-- TrainingRequest.java
|   |-- png.image.Analyzer
|   |   |-- bin
|   |   |   `-- (...)
|   |   |-- lib
|   |   |   `-- (...)
|   |   |-- resources
|   |   |   `-- services.xml
|   |   |-- run.as.local.sh
|   |   `-- src
|   |       `-- org
|   |           `-- xplico
|   |               |-- plugin
|   |               |   |-- XplicoPlugin_interface.java
|   |               |   `-- XplicoPlugin_main.java
|   |               `-- pluginmanager
```

```
|   |                   |-- adaptionLayer
|   |                   |   |-- local
|   |                   |   |   `-- LocalPluginAdaptionLayer.java
|   |                   |   `-- remote
|   |                   |       |-- PgCallbackHandler.java
|   |                   |       |-- PgMessageReceiverInOut.java
|   |                   |       `-- PgSkeleton.java
|   |                   `-- common
|   |                       |-- AnalysisRequest.java
|   |                       |-- PluginResponse.java
|   |                       `-- TrainingRequest.java
|   `-- spamassassin
|       |-- bin
|       |   `-- (...)
|       |-- lib
|       |   `-- (...)
|       |-- resources
|       |   `-- services.xml
|       |-- run.as.local.sh
|       `-- src
|           `-- org
|               `-- xplico
|                   |-- plugin
|                   |   |-- XplicoPlugin_interface.java
|                   |   `-- XplicoPlugin_main.java
|                   `-- pluginmanager
|                       |-- adaptionLayer
|                       |   |-- local
|                       |   |   `-- LocalPluginAdaptionLayer.java
|                       |   `-- remote
|                       |       |-- PgCallbackHandler.java
|                       |       |-- PgMessageReceiverInOut.java
```

```
|                       |       `-- PgSkeleton.java
|                       `-- common
|                           |-- AnalysisRequest.java
|                           |-- PluginResponse.java
|                           `-- TrainingRequest.java
|-- readme.txt
|-- src
|   |-- build.sh
|   |-- db
|   |   |-- dbOperation.cpp
|   |   `-- dbOperation.h
|   |-- pm.client
|   |   |-- pm_applicationLayer.cpp
|   |   |-- pm_databases.cpp
|   |   |-- pm_databases.h
|   |   |-- pm_logicLayer.cpp
|   |   |-- pm_logicLayer.h
|   |   |-- pm_messages.cpp
|   |   |-- pm_messages.h
|   |   |-- pm_transportLayer.cpp
|   |   |-- pm_transportLayer.h
|   |   |-- pm_transportLayer_local.cpp
|   |   |-- pm_transportLayer_local.h
|   |   |-- pm_transportLayer_remote.cpp
|   |   |-- pm_transportLayer_remote.h
|   |   |-- pm_utils.cpp
|   |   `-- pm_utils.h
|   |-- pm.server.plugins
|   |   |-- bin
|   |   |-- lib
|   |   |   `-- (...)
|   |   |-- resources
```

```
|   |   |   `-- services.xml
|   |   `-- src
|   |       `-- org
|   |           `-- xplico
|   |               |-- plugin
|   |               |   |-- XplicoPlugin_interface.java
|   |               |   `-- XplicoPlugin_main.java
|   |               `-- pluginmanager
|   |                   |-- adaptionLayer
|   |                   |   |-- local
|   |                   |   |   `-- LocalPluginAdaptionLayer.java
|   |                   |   `-- remote
|   |                   |       |-- PgCallbackHandler.java
|   |                   |       |-- PgMessageReceiverInOut.java
|   |                   |       `-- PgSkeleton.java
|   |                   `-- common
|   |                       |-- AnalysisRequest.java
|   |                       |-- PluginResponse.java
|   |                       `-- TrainingRequest.java
|   `-- tomcat.embedded
|       |-- bin
|       |   `-- (...)
|       |-- doc
|       |   `-- (...)
|       |-- lib
|       |   `-- (...)
|       `-- src
|           `-- ws
|               `-- server
|                   `-- PluginServer.java
`-- test
    `-- (...)
```

## 4.2. Implementación de Interfaces de comunicación intermodular

- **Interfaz ofrecida por el PluginManager a Xplico**

  Xplico podrá utilizar toda la funcionalidad ofrecida por el PluginManager a través del interfaz *'pm_ logicLayer.h'*, el cual contiene los siguientes métodos:

```
/*************************************************************************
*  DESCRIPTION: Constructor de la clase. Inicializa PluginManager Logic Layer.
*  INPUT:  @param callback, puntero a función callback (Xplico).
*          @param pmDB, PluginManager Database.
*  OUTPUT: (none)
*************************************************************************/
pm_logicLayer (void (*xplico_callback_func)
                    (string contentPath, int contentPriority,
                     string contentDescription, void *callbackData),
               pm_databases pmDB);
/*************************************************************************
*  DESCRIPTION: Modifica el puntero a función de callback (Xplico).
*  INPUT:  @param NEW_xplico_callback_func, puntero a función callback.
*  OUTPUT: (none)
*************************************************************************/
void setCallbackFunction (void (*NEW_xplico_callback_func)
                                (string contentPath, int contentPriority,
                                string contentDescription, void *callbackData));
/*************************************************************************
*  DESCRIPTION:  Lanza una nueva solicitud de procesamiento (Análisis).
*  INPUT:  @param xaReq, información para el análisis.
*          @param xplicoContext, contexto de Xplico.
*  OUTPUT: (none)
*************************************************************************/
void analyzeContent (XPLICO_ANALYSIS_REQUEST *xaReq, void *xplicoContext);
```

```
/*************************************************************************
*  DESCRIPTION:  Lanza una nueva solicitud de entrenamiento.
*  INPUT:  @param xtReq, información para el entrenamiento.
*          @param xplicoContext, contexto de Xplico.
*  OUTPUT: (none)
*************************************************************************/
void trainContent (XPLICO_TRAINING_REQUEST *xtReq, void *xplicoContext);
/*************************************************************************
*  DESCRIPTION: Función que "PluginManager Transport Layer" utilizará como
*               función de callback para devolver la respuesta.
*  INPUT:  @param PluginResponse, respuesta generada por el Plugin.
*          @param PLUGINMANAGER_CONTEXT, contexto del PluginManager.
*  OUTPUT: (none)
*************************************************************************/
static void pm_callback (PLUGIN_RESPONSE *pluginResponse,
                         PLUGINMANAGER_CONTEXT *pmContext);
```

A continuación analizaremos cada una de estas funciones, especificando sus parámetros de entrada. Para ello recordemos que se trata de un API asíncrono, de modo que aporta la funcionalidad necesaria para ello, es decir la posibilidad de retornar asíncronamente las respuestas a las llamadas realizadas permitiendo el paso de información de contexto.

Este asincronismo se ha conseguido empleando un mecanismo en el cual Xplico deberá especificar la función a la que se le debe llamar cuando una petición ha sido procesada para devolver la respuesta. Esta función se indicará como parámetro del constructor, el cual es un puntero a dicha función.

De modo que, la función a la cual apunta será la que el PluginManager llamará para responder las peticiones de análisis y entrenamiento. La estructura que debe presentar la función que Xplico pase por parámetro al constructor constará de tres parámetros obligatorios y un último opcional:

```
void (*xplico_callback_func)(string contentPath, int contentPriority,
                             string contentDescription, void *callbackData)
```

- Como primer argumento un *'string'* que representará el *'Path'* del contenido.
- Como segundo argumento un entero que representará el valor de la prioridad asociada al contenido como resultado del procesado. Los valores posibles son los ya descritos en el diseño del sistema [-1, 5].
- Como tercer argumento un *'string'* que representará una breve descripción generada por el plugin tras el procesado.
- Como último y cuarto argumento un puntero a *'void'*, donde será devuelta la información de contexto (si la hubo) que Xplico pasó al PluginManager en la petición de procesado realizada.

Una vez inicializada la capa lógica del PluginManager, Xplico podrá utilizar su funcionalidad a través de las funciones de análisis y entrenamiento especificadas en el interfaz. Como podemos ver estas dos funciones cuentan con dos parámetros de entrada, una estructura de datos C++ donde irá la información necesaria para llevar a cabo el procesado del contenido, y un puntero a *'void'* que servirá para que Xplico pueda indicar al PluginManager la información de contexto que desea conservar, y que debe ser devuelta intacta en la respuesta de la misma.

Estas estructuras de datos que requieren las funciones de procesado se encuentran definidas en *'pm_ messages.h'*, y encapsulan la información ya definida anteriormente en el diseño del sistema. Éstas son:

```
struct XPLICO_ANALYSIS_REQUEST {
    string caseCategory;
    string caseID;
    string contentPath;
    string contentName;
    string contentType;
    long contentTimestamp;
    vector<string> relatedContents;
    vector<string> contentAttributes;
};
```

```
struct XPLICO_TRAINING_REQUEST {
     string caseCategory;
     string caseID;
     int contentPriority;
     string contentPath;
     string contentName;
     string contentType;
     string contentDescription;
     long contentTimestamp;
     vector<string> relatedContents;
     vector<string> contentAttributes;
};
```

- **Interfaz ofrecida por el PluginManager a los Desarrolladores de Plugins**

  Tal y como se estableció en la parte de diseño, el PluginManager proporciona a los desarrolladores de plugins un interfaz que éstos deben implementar para hacer posible la interoperabilidad entre el PluginManager y sus Plugins.

  Este interfaz recibe el nombre de *'XplicoPlugin_ interface.java'*, se encuentra definido y está proporcionado por la parte servidora del PluginManager. Es un interfaz síncrono y su estructura viene dada por las siguientes líneas:

```
package org.xplico.plugin;
import org.xplico.pluginmanager.*;
import java.net.MalformedURLException;

public interface XplicoPlugin_interface {
    public PluginResponse processAnalyzeRequest (AnalysisRequest pmRequest)
                                                    throws MalformedURLException;
    public PluginResponse processTrainRequest (TrainingRequest pmRequest)
                                                    throws MalformedURLException;
}
```

Estos métodos, que deben ser implementados por el Plugin, presentan como parámetros de entrada unas estructuras de datos que contienen toda la información necesaria para que los plugins sean capaces de procesar el contenido. Estas estructuras son generadas a partir de la petición de procesado o entrenamiento lanzada por Xplico, en la capa de adaptación del modulo servidor del PluginManager, que como ya comentamos se encuentra del lado de los Plugins. Además retornan otra estructura java que contiene toda la información referente al resultado provocado tras el proceso.

A continuación se presentarán los atributos de cada una de las clases que representan estas estructuras de datos comentadas. Estos campos de datos serán los mismos que los que se especificaron en el diseño del sistema, de modo que únicamente los recordaremos, sin pararnos a detallar la descripción de cada uno de ellos.

```
/************************ class AnalysisRequest ************************/
String caseCategory;    // provided by Xplico
String caseID;          // provided by Xplico
URL contentURI;         // provided by Xplico
byte[] contentHash;     // calculated by PluginManager
String contentName;     // provided by Xplico
String contentType;     // provided by Xplico or calculated by PluginManager
long contentLength;     // calculated by PluginManager
long contentTimestamp;  // provided by Xplico
URL[] relatedContents;  // provided by Xplico
Properties contentAttributes; // provided by Xplico

/************************ class TrainingRequest ************************/
String caseCategory;    // provided by Xplico
String caseID;          // provided by Xplico
int contentPriority;    // provided by Xplico
URL contentURI;         // provided by Xplico
byte[] contentHash;     // calculated by PluginManager
String contentName;     // provided by Xplico
String contentType;     // provided by Xplico or calculated by PluginManager
```

```
String contentDescription     // provided by Xplico
long contentLength;           // calculated by PluginManager
long contentTimestamp;        // provided by Xplico
URL[] relatedContents;        // provided by Xplico
Properties contentAttributes; // provided by Xplico

/*********************** class PluginResponse ************************/
URL contentURI;
byte[] contentHash;
String contentName;
long contentLength;
long processTimestamp;
int contentPriority;
String processMessage;
URL[] relatedContents;
Properties contentAttributes;
```

Estas clases java que representan el modelo de información de cara a los Plugins cuentan además con sus correspondientes métodos *'get'* y *'set'* por cada atributo.

## 4.3. Implementación Modelo de información del PluginManager

En esta sección veremos la forma en la que se ha implementado el Modelo de información del PluginManager, es decir, la estructura que tiene su base de datos (tablas, campos y tipos). Para ello se ha utilizado la herramienta *'SQLite Database Browser'* que permite explorar bases de datos de tipo SQLite.

En la Figura 4.1 se puede apreciar el aspecto que presenta la base de datos del PluginManager, en concreto se presenta la Tabla que refleja el registro de Plugins. Cada registro en la base de datos se compone de un identificador numérico que asocia la propia base de datos directamente al completarse el registro, el nombre del plugin, el tipo de contenidos que analiza, la dirección donde se localiza, y una breve descripción del mismo. Seguida a esta, en la Tabla 4.4 se describe cada uno de los campos del registro y el tipo asociado al mismo.

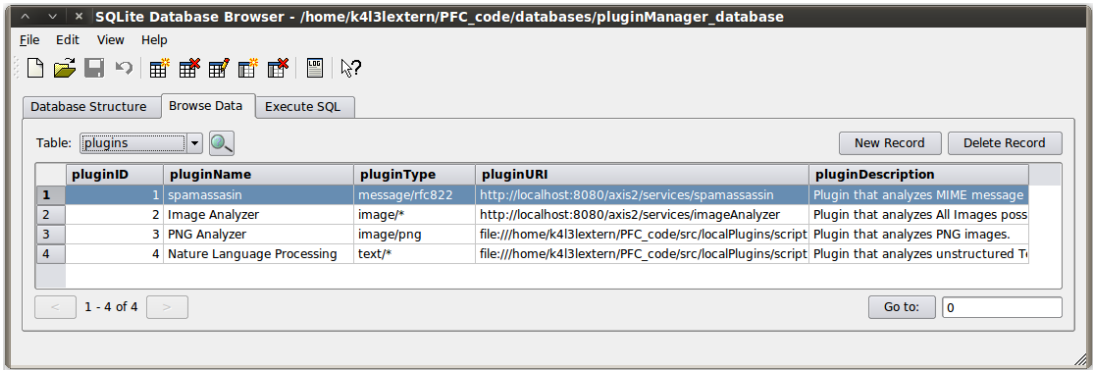


Figura 4.1: Base de datos PluginManager: Implementación del Registro de Plugins (SQLite).

| **Campo** | **Descripción** | **Tipo** |
|---|---|---|
| pluginID | Identificador del plugin en el registro. | INTEGER PRIMARY KEY |
| pluginName | Nombre asociado al Plugin. | TEXT |
| pluginType | Tipo de contenidos que analiza. | TEXT |
| pluginURI | Ubicación del Plugin. | TEXT |
| pluginDescription | Descripción del Plugin. | TEXT |

Tabla 4.4: *Base de datos PluginManager: Implementación del Registro de Plugins.*

En la Figura 4.2 se ve reflejado el aspecto de la Tabla que contempla las listas de procesado, es decir, las relaciones entre tipos de contenidos que pueden ser analizados y los plugins disponibles capaces de procesar ese tipo de contenidos. Cada entrada corresponde a una regla compuesta por tres campos, que son, un identificador de plugin, un orden de ejecución y el tipo de contenido a analizar. De modo que, por ejemplo si tenemos un correo electrónico de tipo *'message/rfc822'* para procesar, el PluginManager consultará esta tabla para saber la secuencia de plugins a llamar. Tal y como muestra la tabla de la figura, en este caso para el tipo *'message/rfc822'* se dispone de dos Plugins cuyos identificadores son, *'1'* y *'4'*, que mirando en la tabla de registro de Plugins corresponden con el *'spamassassin'* y *'Nature Language Processing'* respectivamente. Mirando el orden de ejecución el PluginManager primero lanzará una petición al *'spamassassin'* y si su resultado es negativo (prioridad asociada con valor de pre-clasificación desconocida ('0') o de error ('-1')) se pasará a realizar una segunda llamada, esta vez al plugin *'Nature Language Processing'*.

Además de disponer de la posibilidad de configurar una lista de ejecución, la lógica interna del Plugin sigue un mecanismo tal que, si después de haber completado toda la lista de ejecución configurada no se ha obtenido una pre-clasificación, volverá a consultar esta tabla en busca de plugins de ámbito menos específico, es decir, de plugins globales que sean capaces de procesar contenidos de todo tipo. Por ejemplo, imaginemos que Xplico ordena analizar una imagen de tipo *'image/png'*. Ahora el PluginManager obtendrá una lista de procesamiento con un solo Plugin, ya que para el tipo *'image/png'* sólo se encuentra una entrada en la base de datos. Al realizar esta llamada no se obtiene una respuesta valida de pre-clasificación. De este modo el PluginManager siguiendo su lógica de procesado trata de realizar una segunda consulta en busca de un Plugin más general, buscando a partir de un tipo menos específico, *'image/*'*. Encontraría un Plugin disponible y volvería a lanzar peticiones. Por último, en caso de seguir sin obtener un resultado de pre-clasificación adecuado, trataría de realizar la última consulta, aún menos restrictiva, de ámbito global, es decir, una consulta al tipo *'*'* (Registro de un Plugin capaz de analizar cualquier tipo de contenido). Como podemos ver en la figura adjuntada, en ese momento no se disponía de ningún plugin de ámbito global, de modo que ese contenido se quedaría sin poder pre-clasificar. En caso de haber dispuesto de tal plugin, el PluginManager haría un último lanzamiento de petición.

En resumidas cuentas, la secuencia de consultas a la base de datos para la obtención de las direcciones de Plugins de procesado para un contenido específico (en el caso del análisis de un correo electrónico por ejemplo) son las siguiente:

1. Obtención de la lista de procesado de plugins específicos: 'message/rfc822'

```
"SELECT plugins.pluginURI
FROM typesPlugins,plugins
WHERE typesPlugins.type=='message/rfc822'
AND typesPlugins.pluginID==plugins.pluginID
ORDER BY typesPlugins.pluginOrder;"
```

2. Obtención de Plugin de procesado sub-específico: 'message/*'

```
"SELECT plugins.pluginURI
FROM typesPlugins,plugins
WHERE typesPlugins.type=='message/*'
AND typesPlugins.pluginID==plugins.pluginID
ORDER BY typesPlugins.pluginOrder;"
```

3. Obtención de Plugin de procesado de carácter global: '*'

```
"SELECT plugins.pluginURI
FROM typesPlugins,plugins
WHERE typesPlugins.type=='*'
AND typesPlugins.pluginID==plugins.pluginID
ORDER BY typesPlugins.pluginOrder;"
```

En la Tabla 4.6 se describe cada uno de los campos de cada regla y el tipo asociado al mismo.

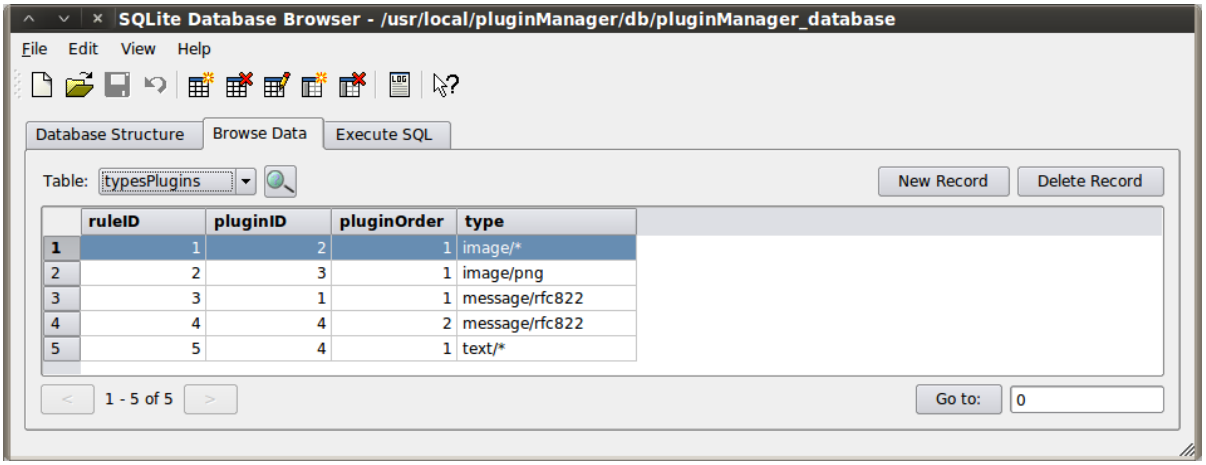


Figura 4.2: Base de datos PluginManager: Implementación de Listas de operación (SQLite).

| Campo | Descripción | Tipo |
|---|---|---|
| ruleID | Identificación de la regla. | NUMERIC |
| pluginID | Identificación del Plugin. | NUMERIC |
| type | Tipo de contenido a analizar. | TEXT |
| pluginOrder | Orden de ejecución del Plugin para el *'Analysis Type'* en cuestión. | NUMERIC |

Tabla 4.6: *Base de datos PluginManager: Implementación de Listas de operación.*

Por último la Figura 4.3 refleja el aspecto de la caché de clasificaciones empleada por el PluginManager. Esta Caché será consultada antes de obtener la lista de plugins de procesamiento para el contenido indicado, ya que si el contenido ya ha sido clasificado con anterioridad no hace falta lanzar una nueva petición, el valor de clasificación será obtenido de Caché y será directamente devuelto a Xplico junto a su descripción asociada. Cada registro en Caché cuenta con un campo de identificación de contenido generado automáticamente por la base de datos al ser actualizado, un código *'hash md5'*, la longitud del contenido en cuestión, el nombre, la prioridad y la descripción asociadas. Para llevar a cabo las comprobaciones el PluginManager identifica unívocamente un contenido a partir del conjunto de campos:

*'contentHash - contentLength'*

La Tabla 4.8 describe cada uno de los campos de cada entrada en caché y el tipo asociado.

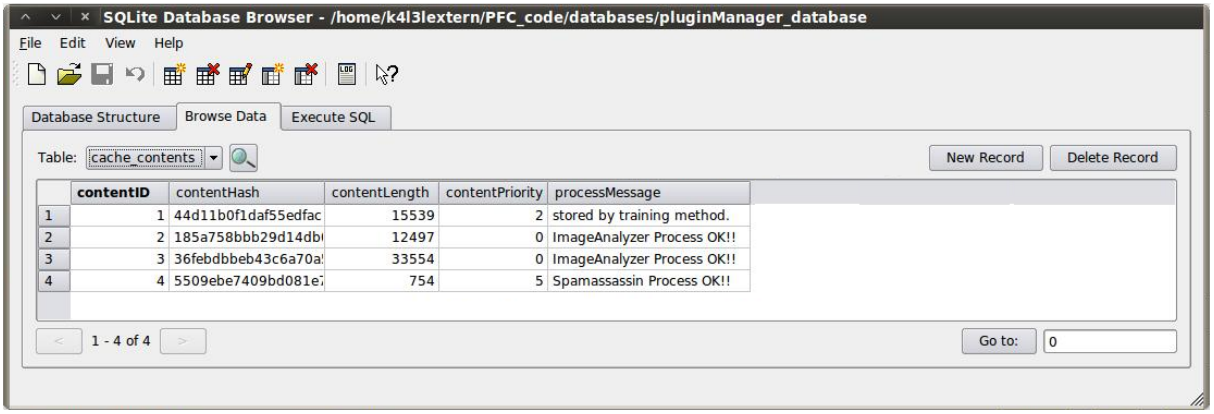


Figura 4.3: Base de datos PluginManager: Implementación de Caché de contenidos (SQLite).

| **Campo** | **Descripción** | **Tipo** |
|---|---|---|
| ContentID | Identificador del contenido. | INTEGER PRIMARY KEY |
| ContentHash | Secuencia Hash que resume el contenido. | TEXT |
| ContentLength | Tamaño del contenido. | NUMERIC |
| ContentPriority | Prioridad asociada al contenido. | NUMERIC |
| ProcessMessage | Descripción asociada al procesado del contenido. | TEXT |

Tabla 4.8: *Base de datos PluginManager: Implementación de Caché de contenidos.*

El PluginManager cuenta con los interfaces *'pm_ databases.h'* y *'db_ operation.h'* para interactuar con la base de datos.

El interfaz *'db_ operation.h'* cuenta con la funcionalidad necesaria para tratar con bases de datos SQLite3 mediante el uso de las librerías proporcionadas por SQLite3. Mientras que *'pm_ databases.h'* utiliza *'db_ operation.h'* para implementar las funciones que necesita el PluginManager para interactuar con su base de datos.
Las funciones que *'db_ operation.h'* aporta son:

```
/*****************************************************************************
*  DESCRIPTION:  Constructor de la clase. Inicialización de la base de datos.
*  INPUT:   @param db_name, Database filename (UTF-8).
*  OUTPUT:   (none)
*****************************************************************************/
dbOperation (const char *db_name);

/*****************************************************************************
*  DESCRIPTION:  Devuelve un conector a la base de datos sqlite3
*                que está siendo usada.
*  INPUT:   (none)
*  OUTPUT:   @return *getDB, Database Connection Object.
*****************************************************************************/
sqlite3* getDB (void) {return db;}

/*****************************************************************************
*  DESCRIPTION:  Devuelve el nombre de la base de datos que está siendo usada.
*  INPUT:   (none)
*  OUTPUT:   @return const char *, Database Name
*****************************************************************************/
const char* getDBName (void) {return dbName;}

/*****************************************************************************
*  DESCRIPTION: Cambio de la base de datos que se está utilizando.
*  INPUT:  @param db, Database Connection Object.
*  OUTPUT:  (none)
*****************************************************************************/
void setDB (sqlite3 *db);
```

```
/*************************************************************************
*  DESCRIPTION:  Cambio del nombre de la base de datos que se usa.
*  INPUT:   @param db_name, Database Name.
*  OUTPUT:   (none)
*************************************************************************/
void setDBName (const char *db_name);

/*************************************************************************
*  DESCRIPTION:  Abre una nueva conexión con la base de datos configurada.
*  INPUT:   (none)
*  OUTPUT:   @return string, (1=Database opened successfully), (0=Error)
*************************************************************************/
int dbOpenConnection(void);

/*************************************************************************
*  DESCRIPTION: Ejecución de comando SQL.
*  INPUT:  @param statement, SQL to be evaluated (Example: SELECT a, b FROM a;)
*  OUTPUT:  @return vector<vector<string>>, represents the result of request.
*************************************************************************/
vector<vector<string> > dbQuery (char *statement);

/*************************************************************************
*  DESCRIPTION: Impresión de comando SQL.
*  INPUT:  @param statements, SQL to be evaluated
*  OUTPUT:  (none)
*************************************************************************/
void print_selectQuery_result (vector<vector<string> > statements);
```

```
/*************************************************************************
*  DESCRIPTION: Devuelve el número de entradas resultantes de una petición.
*  INPUT:  @param statements, SQL to be evaluated
*  OUTPUT:  @return a int that represents the number of entries.
*************************************************************************/
int getNumEntries (vector<vector<string> > statements);

/*************************************************************************
*  DESCRIPTION: Cierra la base de datos.
*  INPUT:  (none)
*  OUTPUT:  (none)
*************************************************************************/
void dbClose(void);
```

El interfaz *'pm_ databases.h'* hace uso del interfaz *'db_ operation.h'* proporcionando funciones para el tratamiento directo con la base de datos del PluginManager. Las funciones aportadas son:

```
/*************************************************************************
*  DESCRIPTION: Constructor de clase. Inicialización de SQLite3 database.
*  INPUT:       @param dbPM,  represents the pluginManager Database Path.
*************************************************************************/
pm_databases (string dbPM);

/*************************************************************************
*  DESCRIPTION: Verifica si el contenido especificado fue ya procesado
*               y se encuentra almacenado en chaché.
*  INPUT:  @param contentHash, código hash del contenido.
*          @param contentLength, longitud del contenido.
*  OUTPUT: @return contentPriority y processMessage como primer y segundo
 elemento del objeto devuleto.
*************************************************************************/
std::pair<long,string> checkCache (string contentHash, long contentLength);
```

```
/*************************************************************************
*  DESCRIPTION: actualiza cache con el contenido indicado.
*  INPUT:       @param contentHash, código hash md5 que resume el contenido.
*               @param contentLength, longitud del contentenido.
*               @param contentPriority, prioridad asociada al contenido.
*               @param contentDescription, descripción asociada al contenido.
*  OUTPUT:      (none)
*************************************************************************/
int updateCache (string contentHash, long contentLength,
                 int contentPriority, string contentDescription);

/*************************************************************************
*  DESCRIPTION: Verifica la lista de plugins disponibles para procesar.
*  INPUT:       @param contentType, tipo del contenido (ej.: message/email).
*  OUTPUT:      @param URI_endpoints, un puntero a vector que contiene la
*               lista de URIs donde se localizan los plugins de procesado.
*               @return resultado numérico y endpoint_uri como primer y
*               segundo elemento del objeto devuleto.
*************************************************************************/
int checkPluginAvailability (string contentType, vector<string> *URI_endpoints);

/*************************************************************************
*  DESCRIPTION: Gets the data of the contents that have not yet processed.
*  INPUT:       (none)
*  OUTPUT:      @param xAR,  vector that contains all XplicoRequests to
*               process the contents.
*               @param xCTX, vector that contains all XplicoContexts
*               associated to the contents vector.
*************************************************************************/
void getContents_fromBBDD (vector<XPLICO_ANALYSIS_REQUEST*> *xAR,
                           vector<XPLICO_CONTEXT*> *xCTX);
```

```
/*************************************************************************
*  DESCRIPTION: updates contents table with the specified content.
*  INPUT:       @param description, description associated to the content.
*               @param priorityAssociated, new priority associated to the
*               content.
*               @param contentName, name of the content.
*               @param contentPath, path of the content.
*               @param contentType, type of the content (mime type).
*  OUTPUT:      (none)
*************************************************************************/
void updateContents (string description, int priorityAssociated,
                     string contentName, string contentPath, string contentType);

/*************************************************************************
*  DESCRIPTION: Closes a Database.
*  INPUT:       (none)
*  OUTPUT:      (none)
*************************************************************************/
void dbClose(void);
```

## 4.4. Implementación del PluginManager

En esta sección se va a comentar la implementación del PluginManager, las rutinas, toma de decisiones, y demás aspectos de desarrollo. Para ello se han elaborado unos diagramas de flujo que muestran de una forma esquemática el proceso que siguen los diferentes módulos que componen el PluginManager a la hora de procesar un contenido, tanto para su análisis como su entrenamiento.

A continuación se presenta un esquema de los diferentes diagramas de flujo que se han diseñado para mostrar la forma de trabajo del PluginManager. Comentaremos los aspectos más relevantes de los diagramas para una mejor comprensión de los mismos, que se encuentran adjuntados al final del capítulo.

- **Modulo Cliente (PluginManager Logic and Transport layer)**
  - **Análisis de un contenido**
    - Figura 4.4 - **Capa lógica:** *Llamada a la función de análisis de contenidos.*
      Este diagrama muestra el proceso llevado a cabo por el PluginManager cuando Xplico llama a la función de análisis. La lógica interna de la aplicación puede desembocar en una llamada a la capa de transporte si el contenido debe y puede ser procesado o en la devolución de una respuesta con valor -1, indicando que el contenido no ha podido ser procesado.
      La clase que implementa esta funcionalidad es: *'pm_logicLayer.cpp'*.
    - Figura 4.5 - **Capa de transporte:** *Llamada de Plugins.*
      Esta rutina representa la funcionalidad de la capa de transporte principal, la cual a partir de la dirección de localización del Plugin (endpoint_URI) que se le ha indicado se decide si la llamada a este Plugin se hará local o remotamente. Esta elección se llevará a cabo en función del protocolo asociado a la URI, es decir, si se trata de *'http://'* el Plugin al que llamar se hará remotamente a través de *'Web Services'*, y si por el contrario se trata de *'file://'* la llamada será local.
      La clase que implementa esta funcionalidad es: *'pm_transportLayer.cpp'*.

- Figura 4.6 - **Capa de transporte:** *Llamada a un plugin remoto.*

  Aquí se muestra la forma en que el PluginManager gestiona a través de la capa de transporte la llamada a un plugin situado en otra máquina. Procesado de la información de entrada, encapsulación de la petición en un contenedor SOAP y su envío.

  La clase que implementa esta funcionalidad es: *'pm_transportLayer_remote.cpp'*.

- Figura 4.7 - **Capa de transporte:** *Llamada a un plugin local.*

  Aquí se muestra la forma en que el PluginManager gestiona a través de la capa de transporte la llamada a un plugin local. Procesado de la información de entrada, y la llamada a través de un comando en consola al que se le pasan todos los datos necesarios para procesar el contenido. La respuesta del Plugin, es decir, el *'contentPriority'* y el *'contentDescription'* se obtiene a través de la salida estándar tras su llamada.

  La clase que implementa esta funcionalidad es: *'pm_transportLayer_local.cpp'*.

- Figura 4.8 - **Capa de transporte:** *Retorno respuesta de plugin remoto.*

  Tratándose de un sistema asíncrono, el PluginManager envía peticiones de procesado a Plugins según Xplico se lo va indicando, siguiendo trabajando sin esperar activamente la respuesta de cada petición. Una vez que los Plugins van terminando de procesar las peticiones responderán al PluginManager, es en este momento cuando se ejecuta la funcionalidad encapsulada en la función de callback de Axis implementada en la Capa de transporte del PluginManager. Este diagrama muestra el proceso que sigue esta función de callback tras la recepción de las respuestas de Plugins remotos.

  A grandes rasgos, su función es construir una estructura con la información de la respuesta recibida y devolverla a la capa lógica para procesarla.

  La clase que implementa esta funcionalidad es: *'pm_transportLayer_remote.cpp'*.

- Figura 4.9 - **Capa lógica:** *Retorno respuesta de plugin a capa lógica.*

  La capa de transporte pasará a la capa lógica la respuesta devuelta por el plugin (local o remoto). Esta respuesta a demás de la pre-clasificación asociada, retornará el Contexto encapsulado en la petición original por el PluginManger. Este

contexto contiene la información relacionada con el contenido que se está tratando. Si la pre-clasificación obtenida por el Plugin es válida la caché y la base de datos se actualizarán, mientras que si por el contrario la respuesta refleja la incapacidad del Plugin de procesar el contenido, se consultará en el Contexto la 'lista de procesado' del contenido, en busca del siguiente Plugin disponible para tratar de enviar una nueva petición de procesado. El proceso se repite hasta lograr obtener una pre-clasificación del contenido, o por el contrario hasta que no se disponga de ningún Plugin más para intentar procesarlo.
La clase que implementa esta funcionalidad es: *'pm_ logicLayer.cpp'*.

- **Entrenamiento de un contenido**

  - Figura 4.10 - **Capa lógica:** *Llamada a la función de entrenamiento de contenido*
    Este diagrama muestra el proceso llevado a cabo por el PluginManager cuando Xplico llama a la función de entrenamiento. En un principio trabaja de la misma forma que la llamada a la función de análisis, con la excepción de que siempre que exista un plugin apropiado para entrenar el contenido se hará directamente sin consultar la caché de contenidos.
    La clase que implementa esta funcionalidad es: *'pm_ logicLayer.cpp'*.

  - Figura 4.11 - **Capa de transporte:** *Llamada de Plugins.*
    Al igual que en la función de análisis de contenidos, esta rutina representa la funcionalidad de la capa de transporte principal, la cual a partir de la dirección de localización del Plugin (endpoint_URI) que se le ha indicado decide si la llamada a este Plugin se hará local o remotamente. El mecanismo de selección se lleva a cabo del mismo modo, a través del protocolo asociado a en la URI indicada.
    La clase que implementa esta funcionalidad es: *'pm_ transportLayer.cpp'*.

  - Figura 4.12 - **Capa de transporte:** *Ejecución de un plugin remoto.*
    Muestra la forma en que el PluginManager gestiona a través de la capa de transporte la llamada a un plugin situado en otra máquina. Procesado de la información de entrada, encapsulación de la petición en un contenedor SOAP y su envío. A diferencia de la petición de Análisis, en la de entrenamiento se añaden dos nuevos

campos, que representa el nuevo valor de clasificación (*contentPriority*) y la nueva descripción (*contentDescription*) asociada al contenido a entrenar.
La clase que implementa esta funcionalidad es: *'pm_transportLayer_remote.cpp'*.

- Figura 4.13 - **Capa de transporte:** *Ejecución de un plugin local.*
  Muestra la forma en que el PluginManager gestiona a través de la capa de transporte la llamada a un plugin situado en la máquina local. Procesado de la información de entrada, y llamada al Plugin a través de un comando en consola al que se le pasan todos los datos necesarios para entrenar el contenido. A diferencia de la petición de Análisis, en la de entrenamiento se añaden dos nuevos campos, que representa el nuevo valor de clasificación (*contentPriority*) y la nueva descripción (*contentDescription*) asociada al contenido a entrenar.
  Igualmente la respuesta, se obtiene a través de salida estándar tras su llamada.
  La clase que implementa esta funcionalidad es: *'pm_transportLayer_local.cpp'*.
- Figura 4.14 - **Capa de transporte:** *Retorno respuesta de plugin remoto.*
  Llamada a al función de Callback de Axis implementada en ésta capa de transporte que opera del mismo modo que vimos para la funcionalidad de Análisis.
  La clase que implementa esta funcionalidad es: *'pm_transportLayer_remote.cpp'*.

- Figura 4.15 - **Capa lógica:** *Retorno respuesta de plugin a capa lógica.*
  La llamada de entrenamiento a Plugins se hace sobre todos aquellos plugins que sean del tipo de contenido a entrenar. Las peticiones de entrenamiento se lanzan desde el principio en paralelo e independientemente y no se hace ninguna comprobación a la vuelta de las respuestas que incite nuevas llamadas de entrenamiento.
  La clase que implementa esta funcionalidad es: *'pm_logicLayer.cpp'*.

- **Modulo Servidor (PluginManager-Plugin Adaptation Layer)**

  - Figura 4.16 - **Capa de adaptación:** *Deserialización y Serialización de información.*
    Este diagrama muestra la secuencia de procesamiento de la capa de adaptación del PluginManager situada en el módulo servidor donde se localizan los Plugins. Encargada de deserializar, adaptar la información de la petición, y serializar la respuesta generada por el Plugin, de modo que ahorre trabajo a los desarrolladores de Plugins.

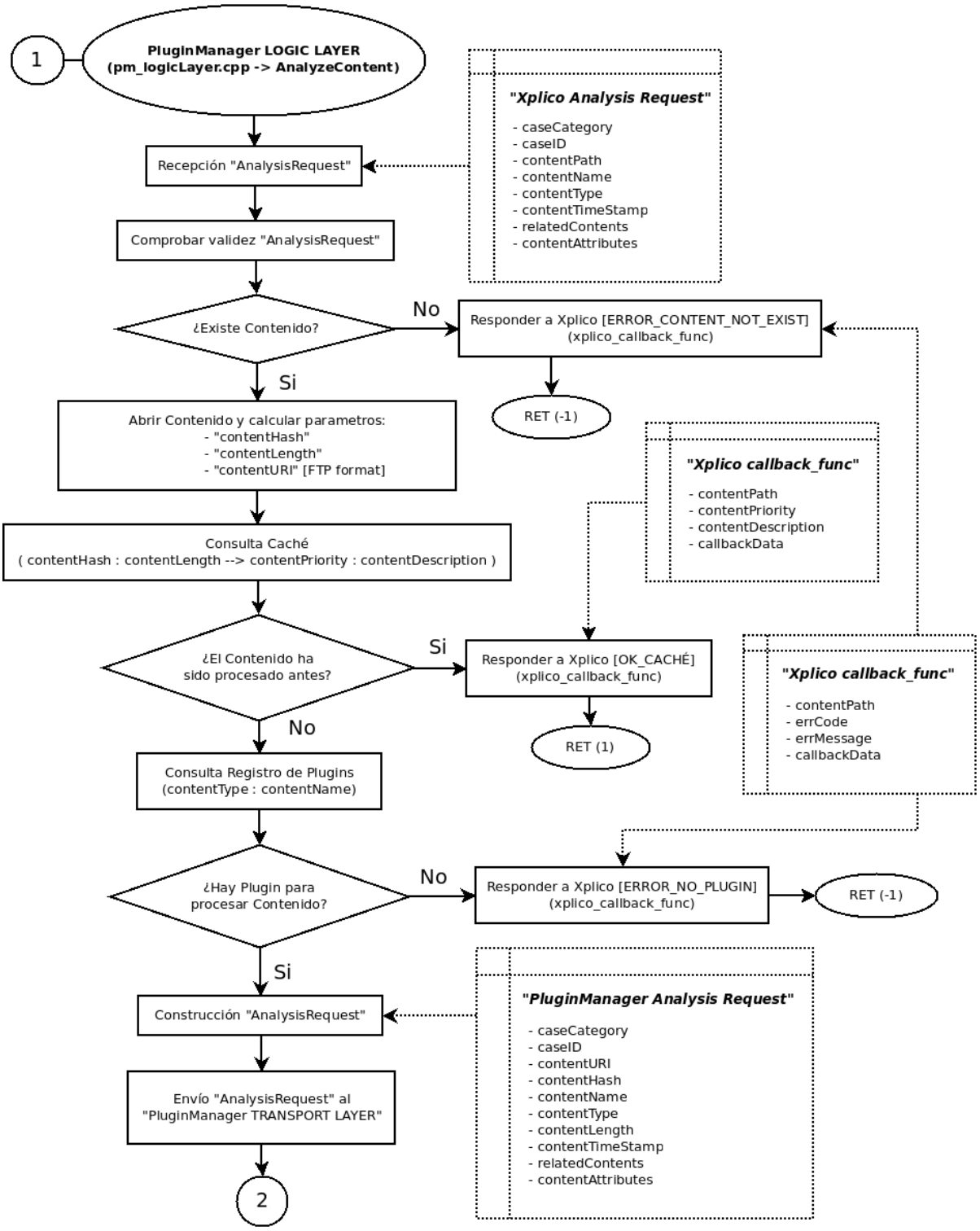


Figura 4.4: Diagrama flujo: PM-Client Analysis (Internal Logic Layer - analyzeContent)

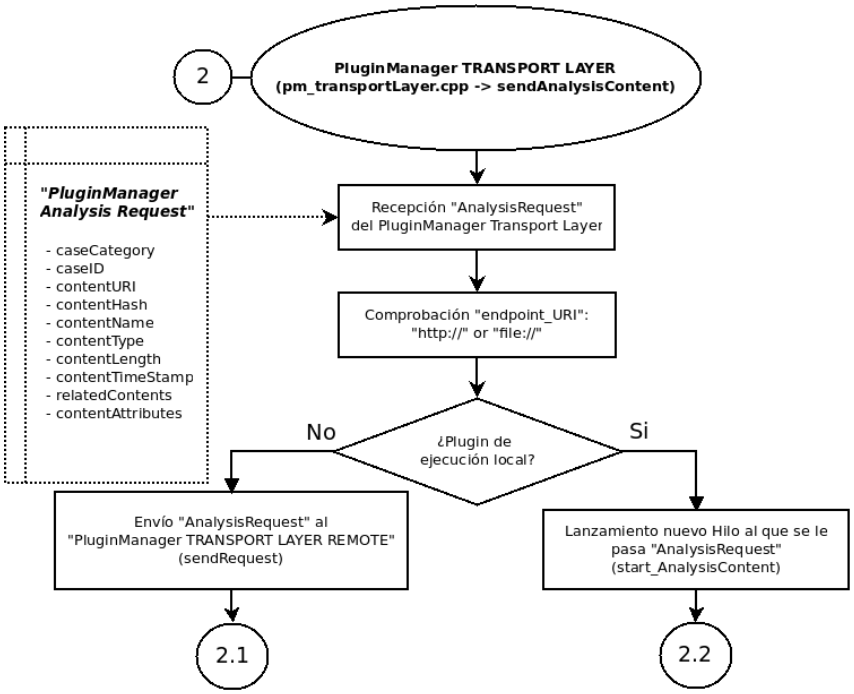


Figura 4.5: Diagrama flujo: PM-Client Analysis (Transport Layer)

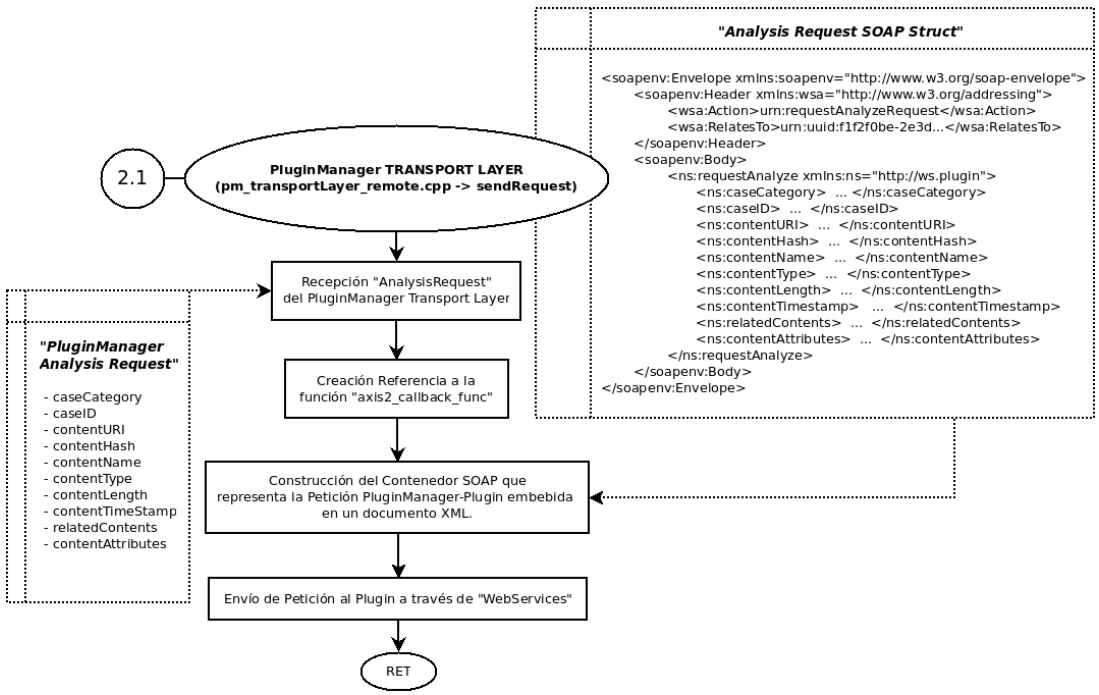


Figura 4.6: Diagrama flujo: PM-Client Analysis (Transport Layer - sendRequest remote plugin)

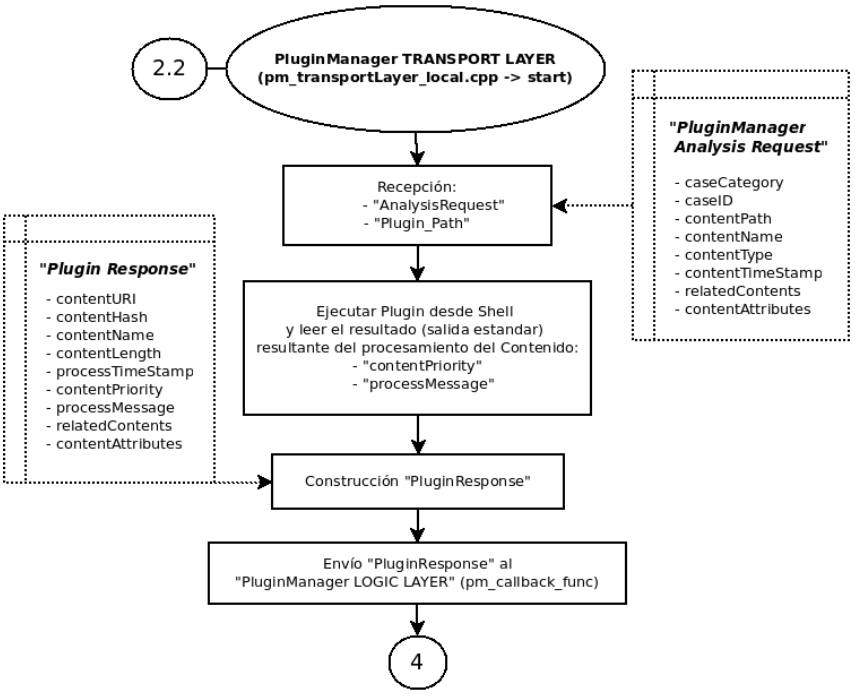


Figura 4.7: Diagrama flujo: PM-Client Analysis (Transport Layer - start thread local plugin)

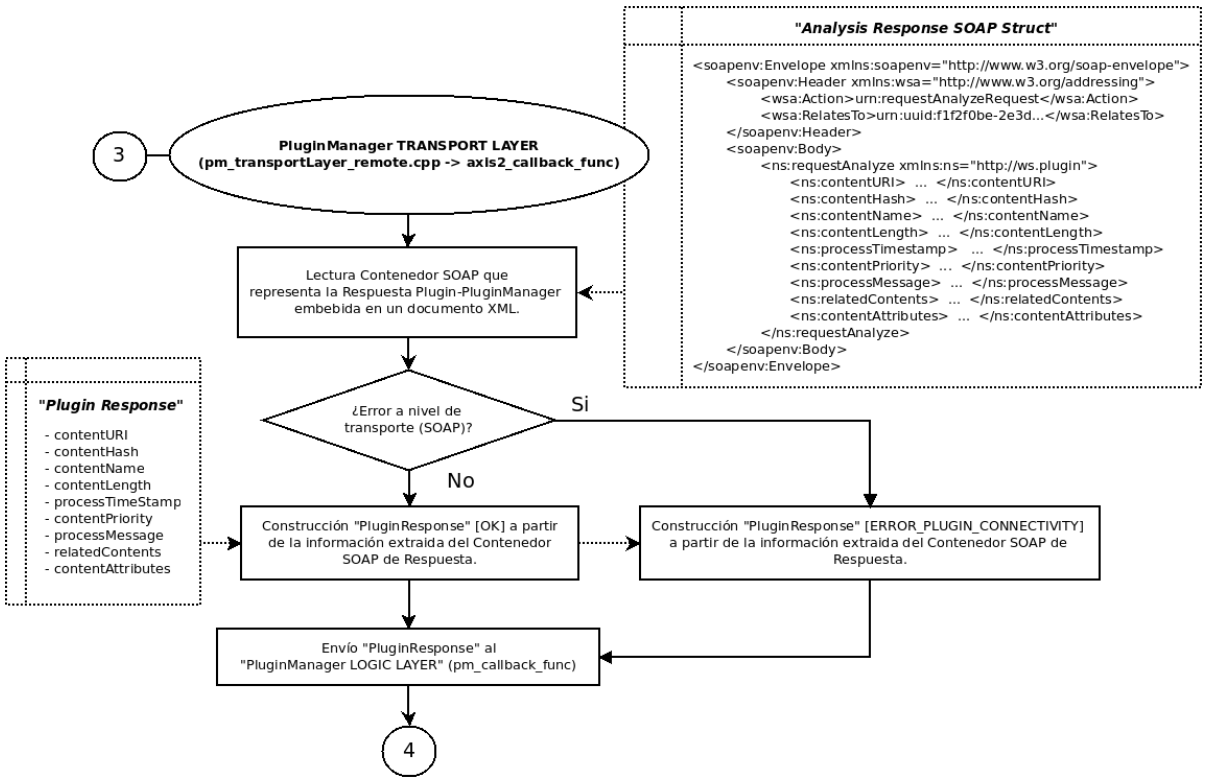


Figura 4.8: Diagrama flujo: PM-Client Analysis (Transport Layer - remote plugin Callback)

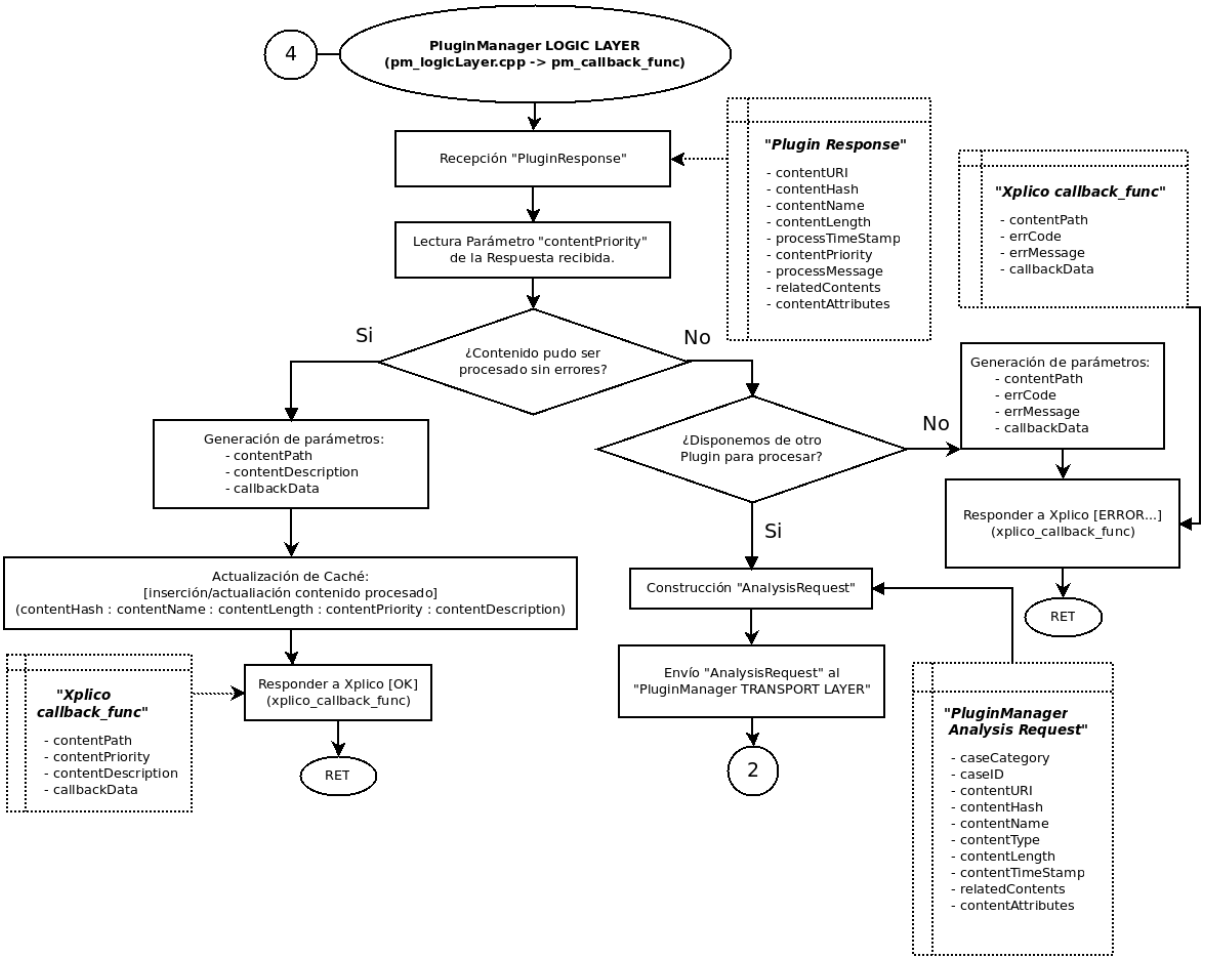


Figura 4.9: Diagrama flujo: PM-Client Analysis (Internal Logic Layer - Transport Callback)

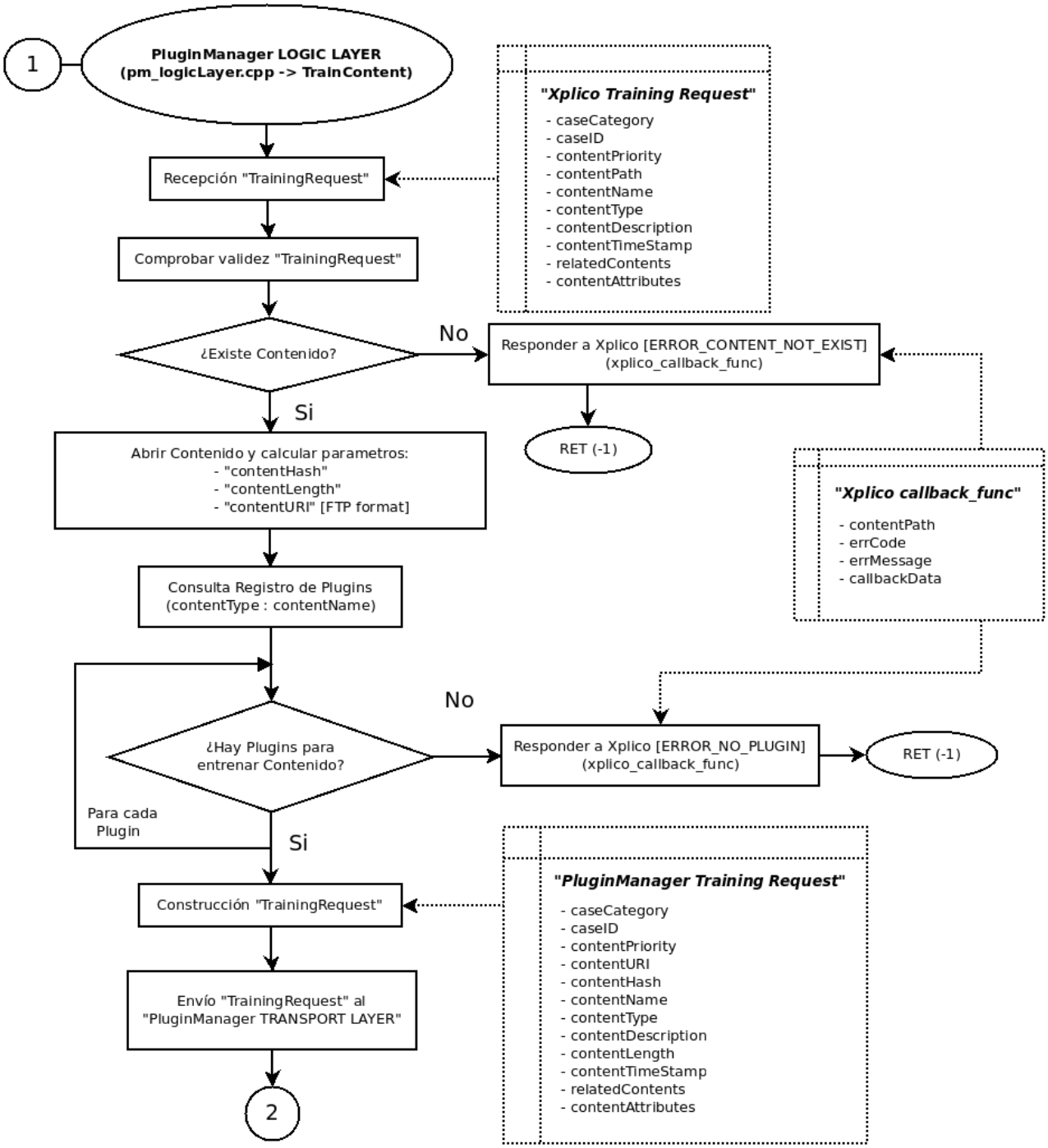


Figura 4.10: Diagrama flujo: PM-Client Training (Internal Logic Layer - trainContent)

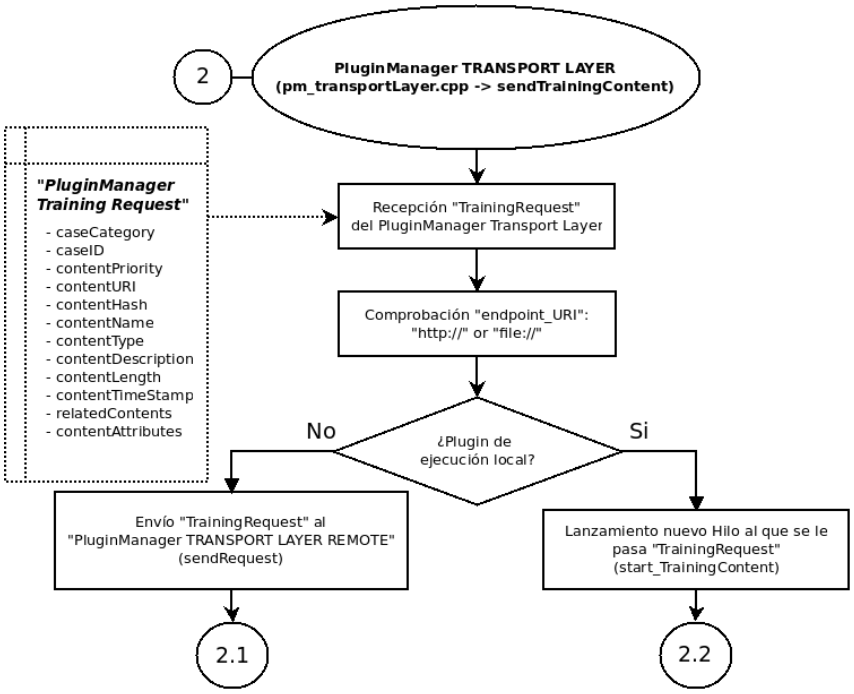


Figura 4.11: Diagrama flujo: PM-Client Training (Transport Layer)

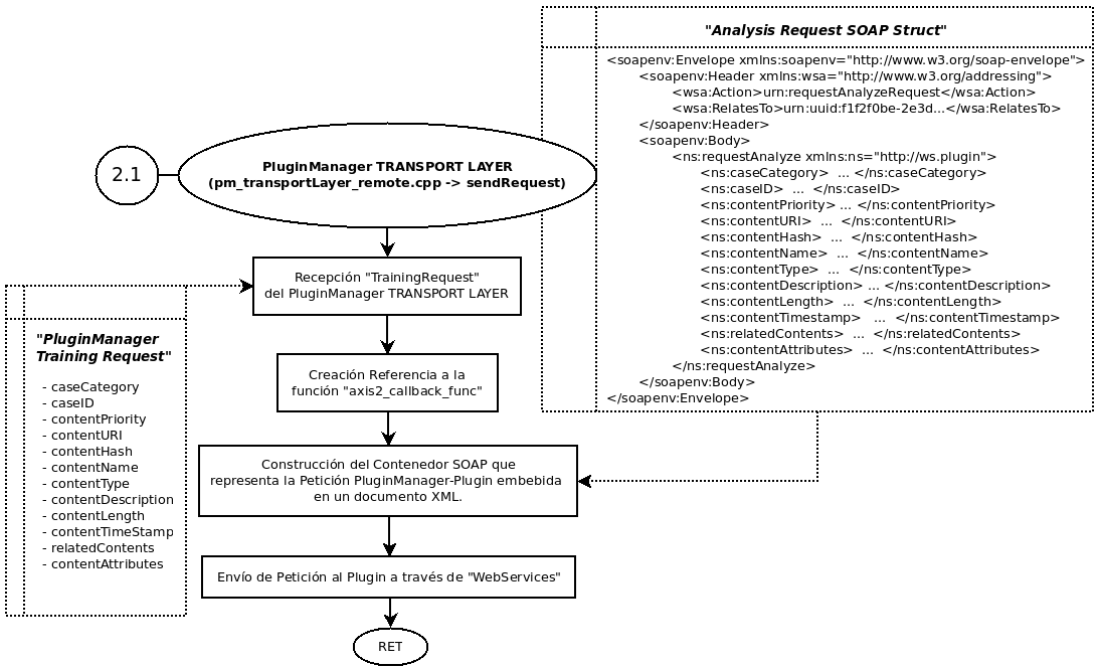


Figura 4.12: Diagrama flujo: PM-Client Training (Transport Layer - sendRequest remote plugin)

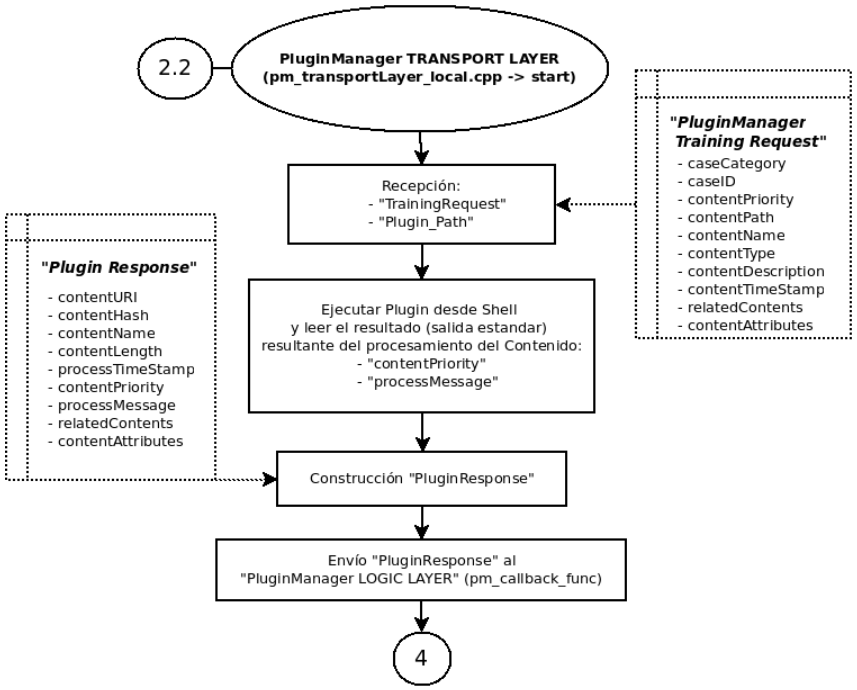


Figura 4.13: Diagrama flujo: PM-Client Training (Transport Layer - start thread local plugin)

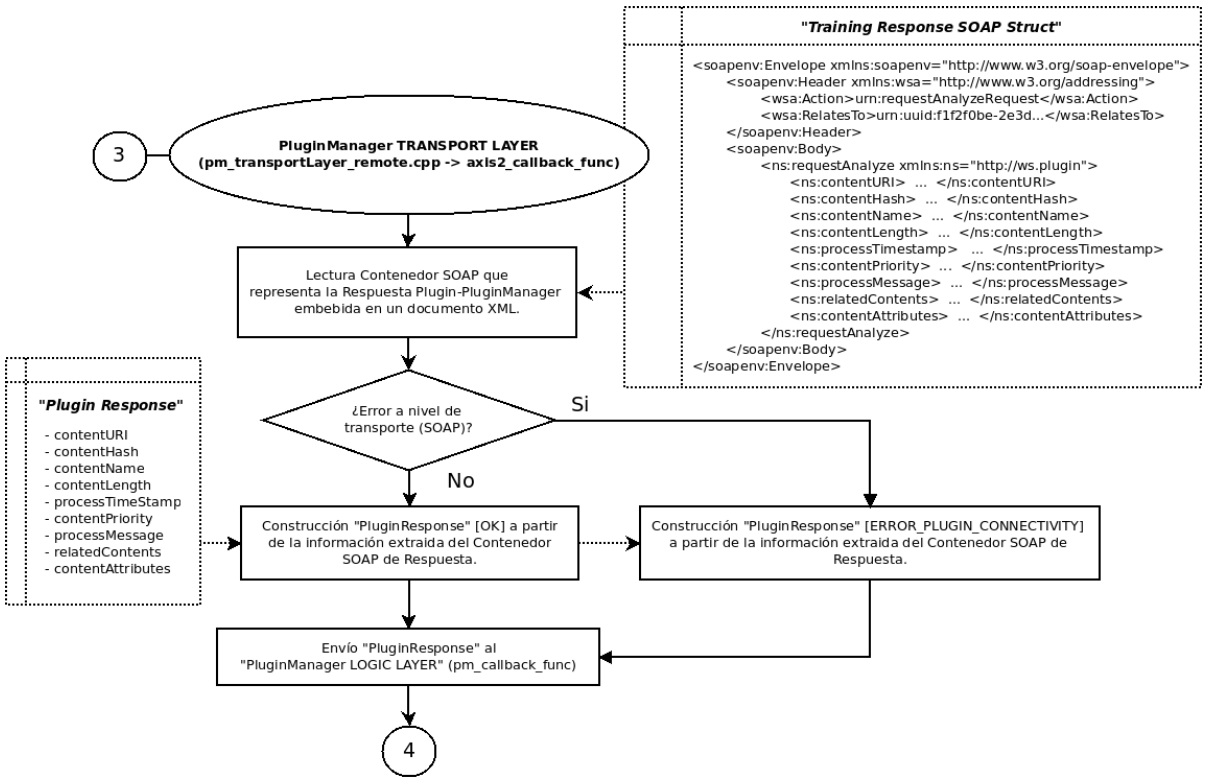


Figura 4.14: Diagrama flujo: PM-Client Training (Transport Layer - remote plugin Callback)

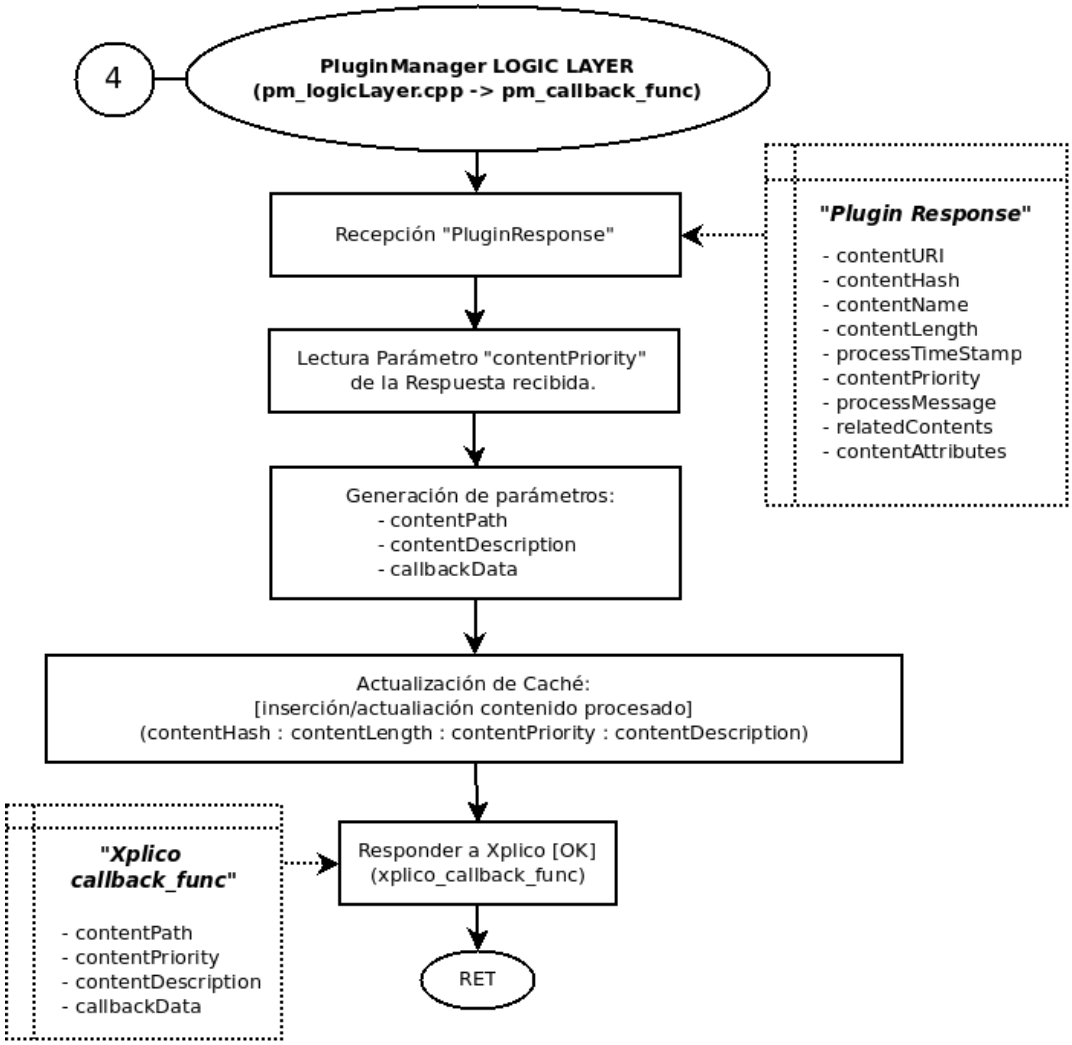


Figura 4.15: Diagrama flujo: PM-Client Training (Internal Logic Layer - Transport Callback)

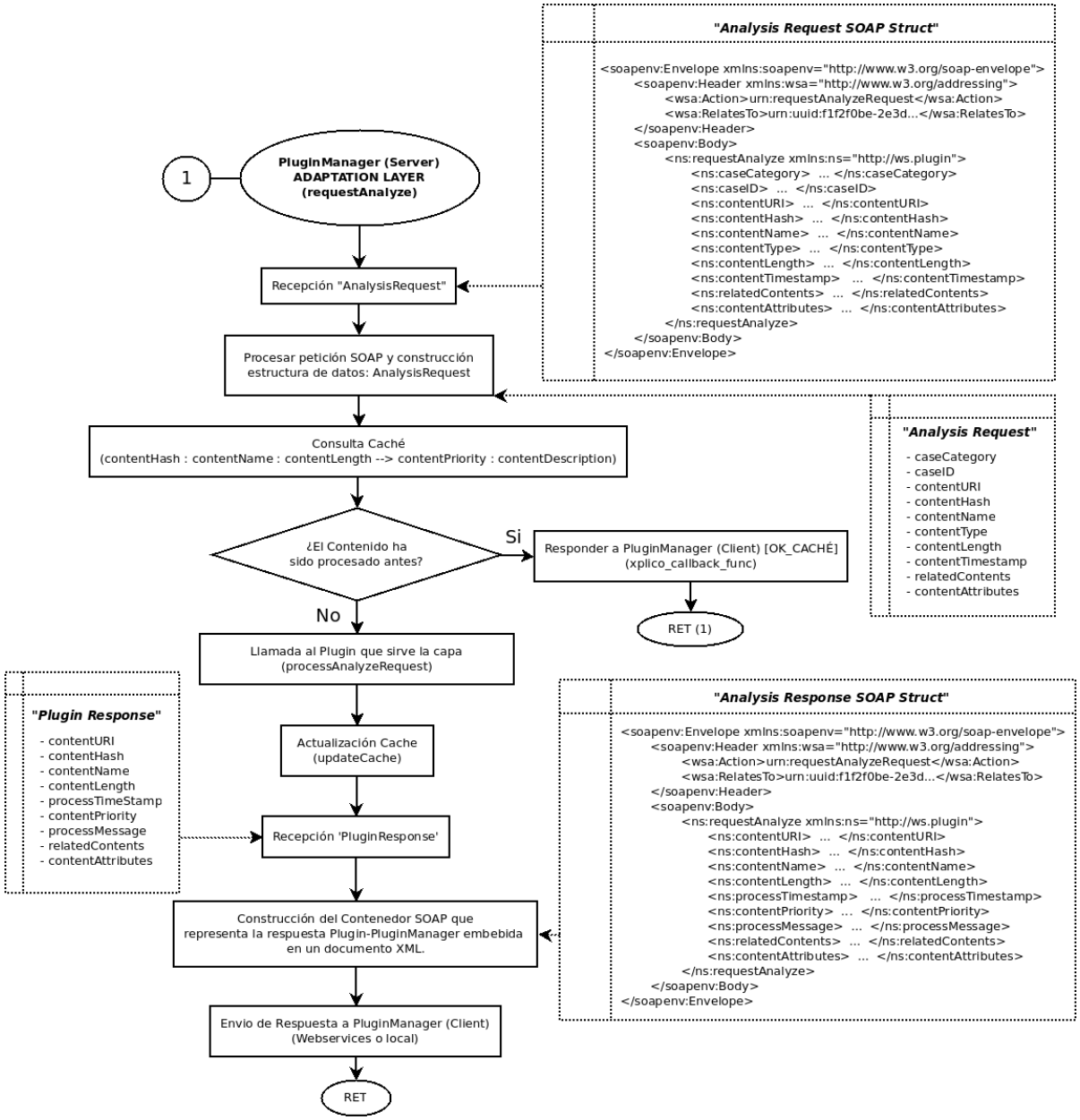


Figura 4.16: Diagrama flujo: PM-Server (Adaptation Layer - requestAnalyze)

## 4.5. Implementación de llamadas a Plugins

- **Llamadas a Plugins Locales**

  Las llamadas a Plugins Locales desde el PluginManager se llevarán a cabo a través de la llamada a un script que realizará la llamada al Plugin localmente una vez realizada la comprobación de la correcta entrada de parámetros para cada función de procesado.

  Este script recibe el nombre de (*'run.as.local.sh'*) y se encuentra situado en el directorio *'root'* de los plugins. Tiene la siguiente forma:

```
if [ "$1"=="analysis" ] && [ $# -eq 9 ]
then
java -classpath ./bin org.xplico.pluginmanager.adaptionLayer.local.
LocalPluginAdaptionLayer "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9"

elif [ "$1"=="training" ] && [ $# -eq 11 ]
then
java -classpath ./bin org.xplico.pluginmanager.adaptionLayer.local.
LocalPluginAdaptionLayer "$1" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9" "$10" "$11"
fi
```

  La llamada realizada desde el código C del PluginManager al script mencionado se hace a través de la siguiente sentencia:

```
utils.callCommandLine(command.str());
```

  Donde el método *'callCommandLine'* de la clase 'utils.cpp' llama al script a través del siguiente método:

```
FILE *pluginCall = popen(command.c_str(), "r" );
```

  Una vez ejecutado el Script y a su vez por éste el comando que llama al Plugin en cuestión, la capa de adaptación del PluginManager que actúa como servidora en el lado del Plugin,

pasará la información a la lógica interna del Plugin y éste, tras procesar el contenido, devolverá una respuesta que la capa de adaptación se encargará de devolver a la parte Cliente del PluginManager. Esta respuesta del plugin es devuelta a través de la salida estándar, dónde la descripción se imprimirá por pantalla y la prioridad asociada al contenido se devolverá como código del resultado de ejecución de la aplicación.

- **Llamadas a Plugins Remotos**

  Como se acordó en la parte de diseño estas llamadas a plugins situados en distintas máquinas que el PluginManager, se lleva a cabo a través de *Web Services*. Para implementarlo se ha utilizado Axis2, en concreto la versión 1.6, la cual se ha obtenido de la página oficial de Axis, http://axis.apache.org/axis2/c/core/download.cgi.

  A continuación comentaremos a grandes rasgos el proceso llevado a cabo por la capa de transporte del PluginManager para posibilitar la comunicación remota con los Plugins a través de *Web Services*.

  - **Inicialización Axis2**

    Como primer paso para el envío de las peticiones desde el PluginManager a Plugins remotos se tiene que configurar el entorno Axis2. Esto lo haremos a través del método implementado de nombre *'initAxis2()'*, el cual configura las siguientes directivas:

    - Configuración del entorno:

      ```
      env = axutil_env_create_all("PlugIns.log", AXIS2_LOG_LEVEL_ERROR);
      ```

    - Configuración de la referencia al end-point:

      ```
      endpoint_ref = axis2_endpoint_ref_create(env, endpoint_uri_axis2_char);
      ```

    - Configuración de opciones:

      ```
      options = axis2_options_create(env);
      axis2_options_set_to(options, env, endpoint_ref);
      axis2_options_set_action(options, env, requestType);
      ```

    - Configuración del client-home (directorio de despliegue):

      ```
      client_home = AXIS2_GETENV("AXIS2C_HOME");
      ```

- Configuración de opciones del cliente Axis2:

```
axis2_svc_client_set_options(svc_client, env, options););
```

- Engranar módulo de direccionamiento:

```
axis2_svc_client_engage_module(svc_client, env, AXIS2_MODULE_ADDRESSING);
```

- **Generación contenedor SOAP**

  Posteriormente se pasará a la creación del contenedor SOAP en el cual irá embebida la información referente a la petición de procesado. Para ello se utilizará el método:

```
axiom_node_t* generateRequestXML (const axutil_env_t *env, axis2_char_t*
                                  request_type, void* request);
```

  Utilizará el entorno inicializado previamente (const axutil_env_t *env), el tipo de petición (analysis o training) y la estructura de datos que representa la petición (XPLICO_ANALYSIS_REQUEST, XPLICO_TRAINING_REQUEST), para generar una referencia a una estructura de datos de tipo *'axiom_node_t'* que representa el contenedor SOAP que se enviará a través de la red.

  *'AnalysisRequest'* y *'TrainingRequest'* son dos estructuras de datos que genera la capa lógica del PluginManager y utiliza en las solicitudes de procesado, pasándolas a la capa de transporte. Su construcción se basa en las estructuras de datos generadas por Xplico (*'XplicoAnalysisRequest'* y *'XplicoTrainingRequest'*) a las que se le han sumado otros campos necesarios para el procesado del contenido como pueden ser el *'contentHash'*, *'contentLength'*, *'contentURI'*, el contexto pasado por Xplico, etc. Al igual que todas las estructuras de datos intercambiadas por el sistema, se encuentran definidas en *'pm_messages.h'* y su contenido es el siguiente:

```
struct ANALYSIS_REQUEST {
    string caseCategory;
    string caseID;
    string contentURI;
    string contentHash;
    string contentName;
    string contentType;
    long contentLength;
    long contentTimestamp;*/
    vector<string> relatedContents; **/
    vector<string> contentAttributes;
}; typedef struct ANALYSIS_REQUEST * AnalysisRequest;

struct TRAINING_REQUEST {
    string caseCategory;
    string caseID;
    int contentPriority;
    string contentURI;
    string contentHash;
    string contentName;
    string contentType;
    string contentDescription;
    long contentLength;/
    long contentTimestamp;
    vector<string> relatedContents;
    vector<string> contentAttributes;
}; typedef struct TRAINING_REQUEST * TrainingRequest;
```

- **Envío del contenedor SOAP a través de la red**

  Por último el envío de la petición embebida en el contenedor SOAP se hace siguiendo los siguientes pasos:

- Inicialización de la estructura de datos Callback.

```
callback = axis2_callback_create(env);
```

- Creación de una referencia a la función de Callback que utilizará Axis para devolver las respuestas a las peticiones.

```
axis2_callback_set_on_complete(callback, plugin_callback_on_complete);
```

- Creación de una referencia a la función de Callback_error a la que llamará Axis para indicar que se ha producido un error en la recepción de la respuesta.

```
axis2_callback_set_on_error(callback, plugin_callback_on_error););
```

- Especificación del Contexto a guardar en la petición.

```
axis2_callback_set_data (callback, get_pmCTX());
```

- Serialización de la petición generando contenedor SOAP.

```
request_node = generateRequestXML(env, requestType, request);
```

- Enviar la petición de procesado al Plugin utilizando el contenedor SOAP generado previamente, el entorno inicializado al principio, e indicando la estructura de callback que hemos configurado dos lineas más arriba.

```
axis2_svc_client_send_receive_non_blocking (svc_client, env,
                                            request_node, callback);
```

Tanto las llamadas a Plugins Locales como a Plugins Remotos se llevan a cabo a través de la Capa de Transporte del PluginManager.

La transición se lleva a cabo desde la Capa Lógica (*'pm_logicLayer.cpp'*) que genera la estructura de datos que representa la solicitud de procesado, pasando por una capa de adaptación de la Capa de Transporte (*'pm_transportLayer.cpp'*) que se encargará de determinar si la llamada al plugin es local o remota. Esta decisión es tomada en base al prefijo del *'endpoint_uri'* proporcionado en su llamada. Si se trata de un endpoint_uri de tipo *'http://'* la llamada será a través de *Web Services*, mientras que si es *'file://'* será local. Esta capa de adaptación en función del tipo de llamada pasará la petición a la capa que procesa las llamadas locales (*'pm_transportLayer_local.cpp'*) o a la que procesa las remotas(*'pm_transportLayer_remote.cpp'*).

## 4.6. Implementación del PluginManager: web GUI

El *Front-end* Web desarrollado como sustituto de Xplico para la presentación del proyecto se encuentra programado en CakePHP, para el cual se ha utilizado la versión 1.3.10, que es la que actualmente se encuentra estable, y podemos obtener de su página oficial a través del siguiente enlace: http://cakephp.org/.

Cuenta con un interfaz inicial que muestra información relativa al PluginManager y puede verse en la Figura 4.17. Contiene enlaces a las diversas plataformas que han intervenido en la realización del proyecto como el proyecto Indect, la herramienta Xplico, la Universidad Carlos III de Madrid, el departamento de Ingeniería Telemática, etc. Además cuenta con información relativa al desarrollador y a los tutores del mismo, y con un enlace directo al *'Home'* de la aplicación.

El interfaz *'Home'* del PluginManager aparece presentado en La Figura 4.18, en la que podemos apreciar que se encuentra estructurado en dos partes:

- Parte izquierda del cuerpo: Menú de configuración del PluginManager.
  - La primera entrada del menú se titula *'Plugins'* (Figura 4.19), y en ella podemos dar de alta y baja plugins en el sistema.
  - La segunda entrada del menú se titula *'Types Plugins'* (Figura 4.20), y sirve para configurar la lista de ejecución de plugins, es decir, el orden de ejecución de plugins para un tipo de contenido concreto.
  - La tercera entrada del menú se titula *'Contents'* (Figura 4.21), y contiene los contenidos insertados en el sistema para ser procesados. Cuenta con funciones de análisis, entrenamiento y edición individual (por contenido).
  - La cuarta y última entrada se titula *'Contents Cache'* (Figura 4.22) y refleja la tabla caché que utiliza el sistema.
- Parte derecha del cuerpo. Sección de análisis de contenidos, que contempla a su vez dos modos de operación.
  - Análisis Global de los contenidos añadidos a través del interfaz de contenidos que muestra la Figura 4.21.

- Análisis Individual: indicar un contenido a través del *'browser'* y pulsar el botón de análisis.

Además cuenta con un sistema para indicar si el PluginManager se encuentra detenido (Imagen Aspa Roja) o si se encuentra en funcionamiento, procesando contenidos (tick verde).

Cabe señalar que según se ha programado este interfaz, la llamada al binario que contiene la funcionalidad del PluginManager al pulsar los botones de análisis y entrenamiento se efectúa a través de la llamada a un script de nombre *'pluginManagerClient.sh'* el cual se encargará de configurar el entorno y finalmente de realizar la llamada al binario PluginManager. La forma que presenta este script de ejecución es la siguiente:

```
#!/bin/sh

  # Axis2 C
  export AXIS2C_HOME=/usr/local/axis2c

  # Axis2/C lib folder to LD_LIBRARY_PATH environment variable
  export LD_LIBRARY_PATH=${AXIS2C_HOME}/lib:$LD_LIBRARY_PATH

  # Start up PluginManager

  case "$1" in
    # Analyze All Contents
    1) /usr/local/pluginManager/bin/pluginManagerClient "$1";;
    # Analyze single content.
    2) /usr/local/pluginManager/bin/pluginManagerClient "$1" "$2" "$3" "$4";;
    # Train content.
    3) /usr/local/pluginManager/bin/pluginManagerClient "$1" "$2" "$3" "$4" "$5" "$6";;
    # Incorrect Execution. So, print help.
    *) /usr/local/pluginManager/bin/pluginManagerClient
  esac
```

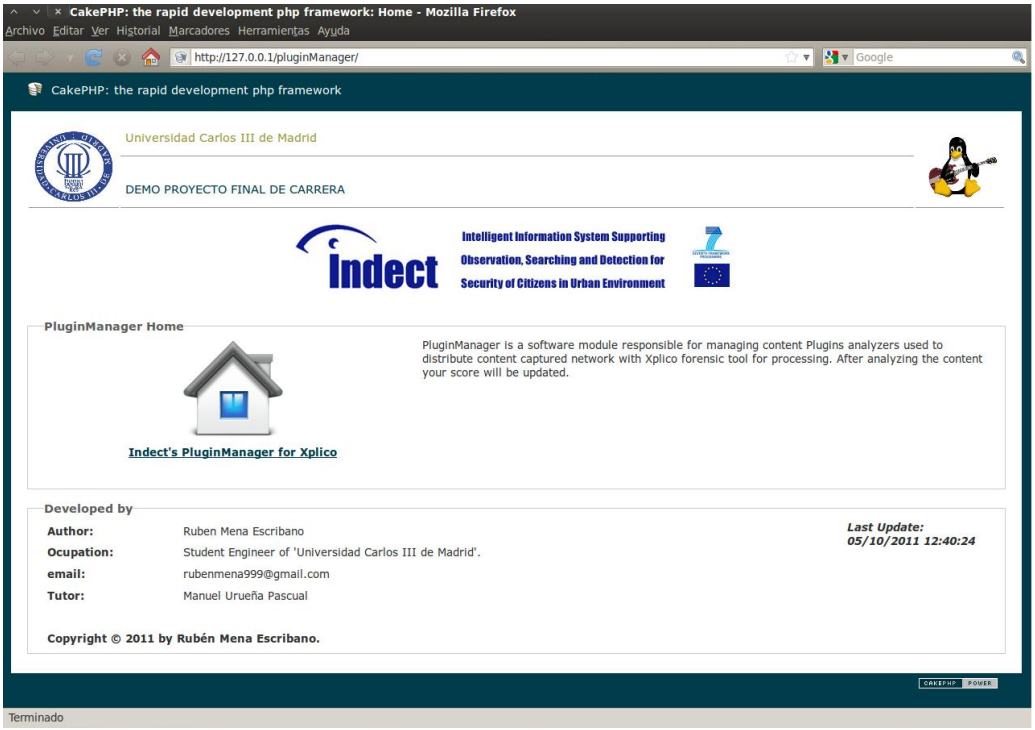


Figura 4.17: PluginManager GUI: Información de la aplicación.

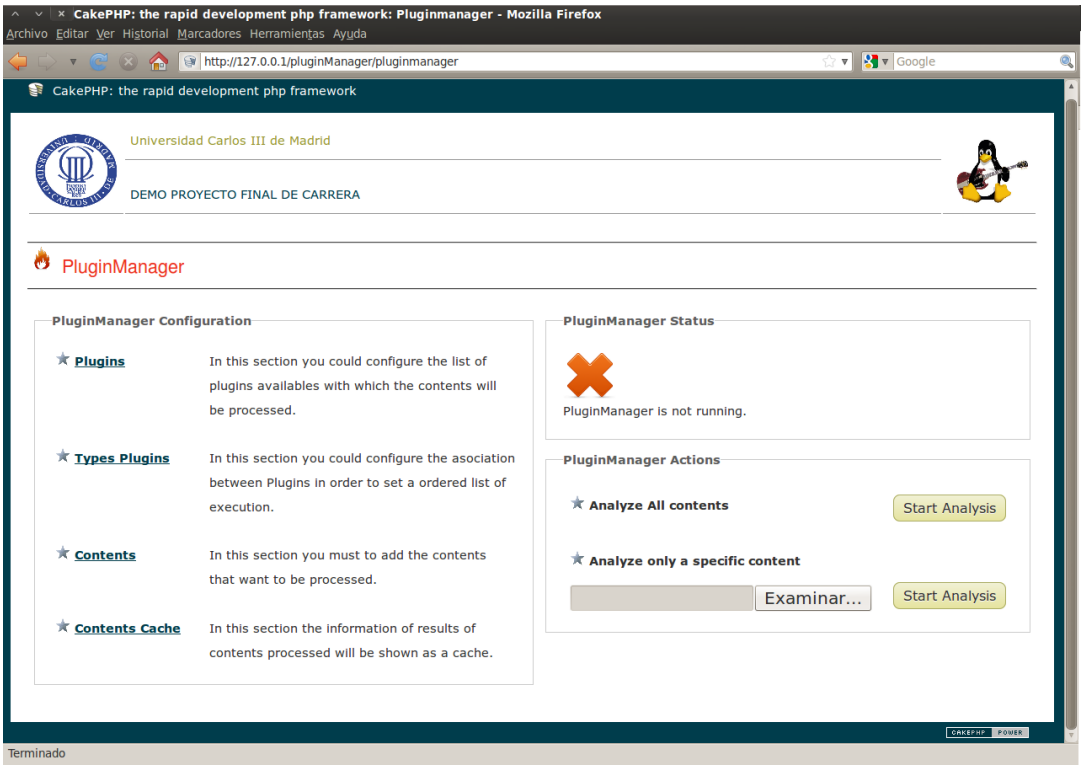


Figura 4.18: PluginManager GUI: PluginManager Home.

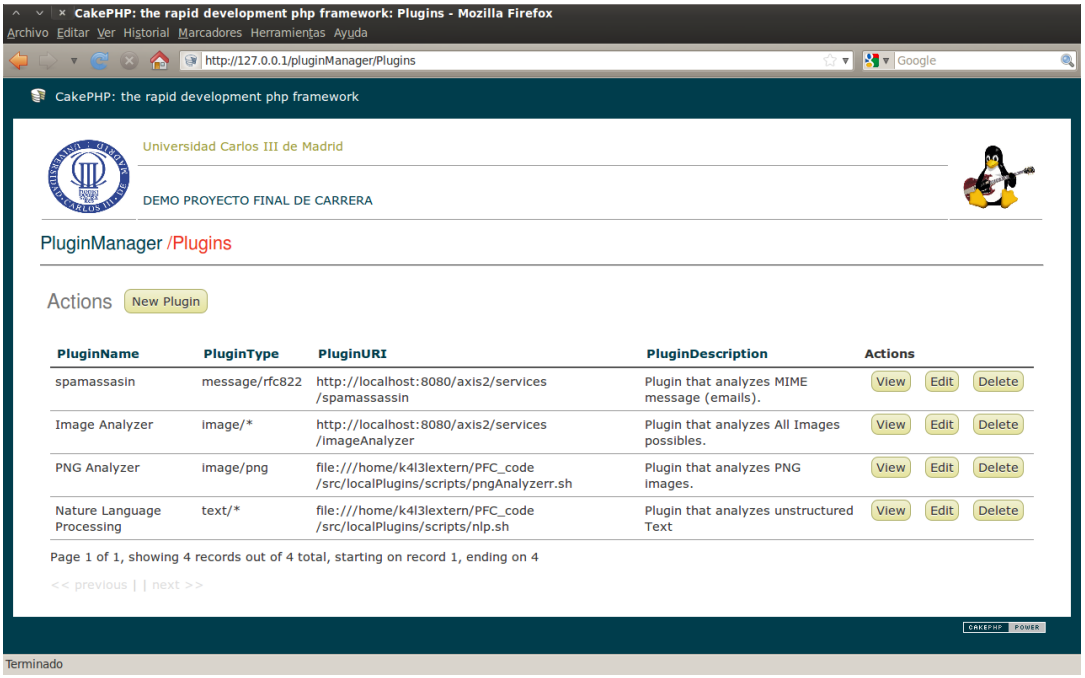


Figura 4.19: PluginManager GUI: Registro de Plugins.

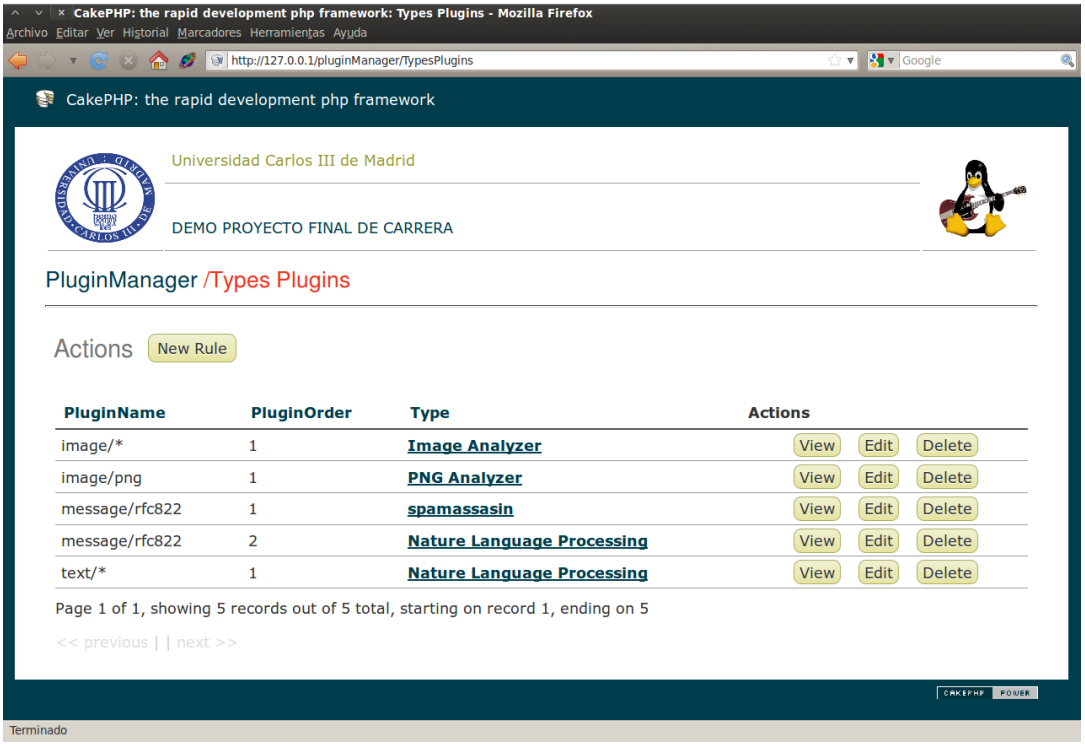


Figura 4.20: PluginManager GUI: Lista de operación.

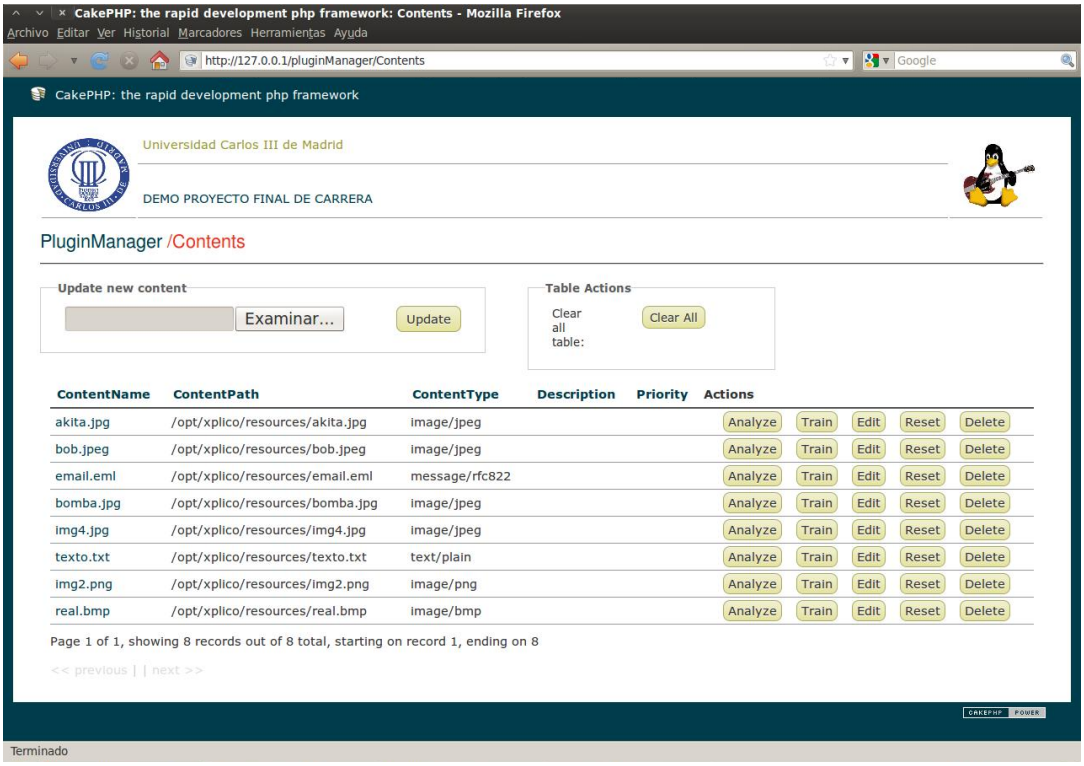


Figura 4.21: PluginManager GUI: Lista de contenidos.

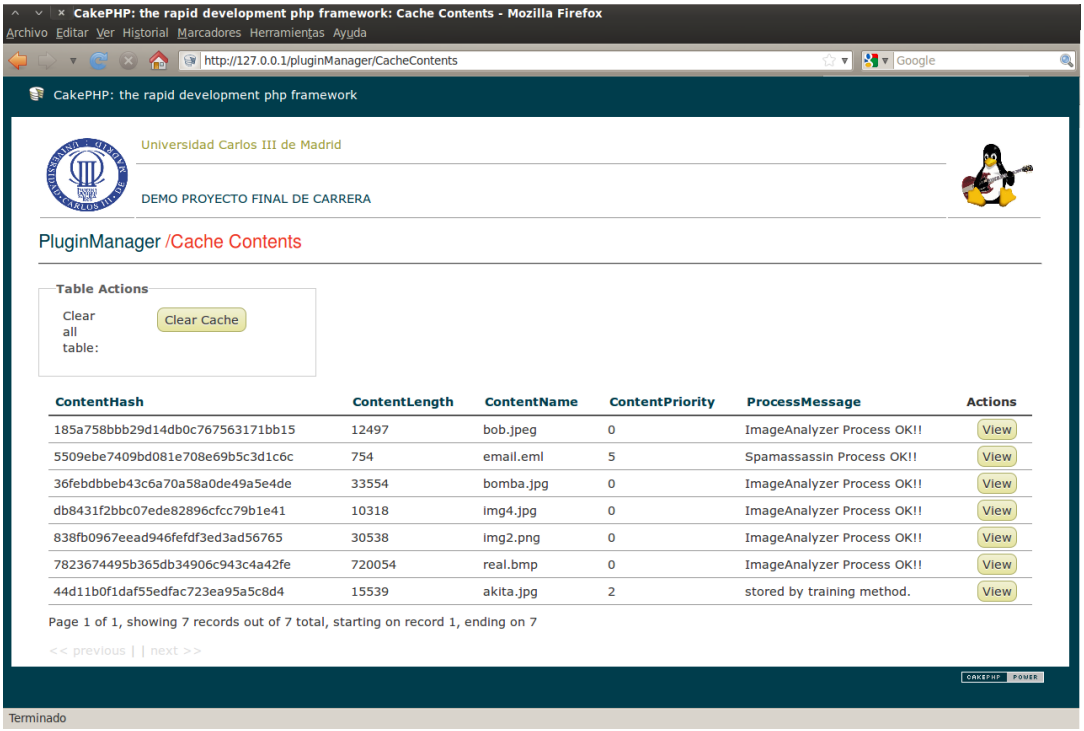


Figura 4.22: PluginManager GUI: Caché de contenidos.

## 4.7. Implementación de Plugins de Ejemplo

Para probar el correcto funcionamiento del PluginManager se han implementado varios Plugins de prueba. En la Tabla 4.10 podemos ver los diferentes Plugins y sus características.

| **Plugin** | **Descripción** |
|---|---|
| Spamassassin | Plugin que determina si un correo electrónico *'message/rfc822'* es Spam. |
| Image Analyzer | Plugin capaz de procesar imágenes de cualquier tipo (*image/**) |
| PNG Analyzer | Plugin que procesa únicamente imágenes de tipo *'image/png'* |

Tabla 4.10: *Plugins de Prueba implementados.*

La lógica interna implementada para cada uno de los Plugins carece de una funcionalidad completa que los dote de la capacidad para procesar los contenidos tal y como se indica en sus descripciones de la Tabla 4.10, a excepción del *'Spamassassin'*. Esto es debido a que el objetivo principal del proyecto no es la elaboración de Plugins, y lo único que se buscaba con la implementación de Plugins de ejemplo era comprobar la validez de los interfaces de comunicación desarrollados y la correcta comunicación intermodular, sin importar la correcta clasificación de los mismos. Sin embargo, en el caso del Spamassassin se ha aprovechado la existencia de una herramienta anti-spam cuyo nombre es igual que el Plugin que la utiliza, y que proporciona la funcionalidad descrita para el mismo. De modo que de los Plugins implementados el Spamassassin es el único que devuelve valores con sentido, acordes a su descripción.

Los Plugins *'Image Analyzer'* y *'PNG Analyzer'* lo único que hacen al llegar una petición, es responder inmediatamente con un valor de prioridad asociada igual a cero. Sin embargo, gracias a la caché implementada por defecto en estos plugins, permitirá probar la funcionalidad de entrenamiento.

La lógica interna del Plugin *'Spamassassin'* al recibir la información, es simplemente la llamada a la herramienta anti-spam de nombre *'Spamassassin'* a la que se le indica la URI del contenido a procesar y devuelve un valor que puede ser, '0' o '1'. El valor '0' especifica que el correo electrónico indicado no es Spam y por el contrario el valor '1' determina que si lo es y por lo tanto se le asocia la prioridad mínima. Una vez terminado el proceso construye la respuesta y

la pasa a la capa de adaptación para que la serialice y la envíe a la parte Cliente del PluginManager.
La herramienta *'Spamassassin'* que utiliza el Plugin desarrollado fue obtenida de:

http://spamassassin.apache.org/downloads.cgi

En cuanto a la función de entrenamiento, esta se encuentra implementada en todos los Plugins de tal modo que actualice su propia caché de contenidos, que no es más que un fichero de texto. Al igual hará la capa de adaptación del PluginManager, la cual cuenta con su propia caché interna.

No entraremos en más detalle de la lógica de los Plugins, ya que ésta es tan simple como se ha descrito. Simplemente hacer referencia al ANEXO C presente al final del documento, en el cual se explica el proceso que se ha seguido para el desarrollo y despliegue de Plugins.

Capítulo 5

# EVALUACIÓN DE LA SOLUCIÓN

Para la evaluación del PluginManager se ha desplegado el sistema completo, que cuenta con una capa de nivel de aplicación desarrollada en C++ que simula a la herramienta Xplico y hace uso del interfaz de comunicación proporcionado por el PluginManager ya comentado, y por otro lado un servidor de Plugins que permite procesar ciertos contenidos. Todo el sistema puede evaluarse a través de la aplicación Web desarrollada con CakePHP que hemos comentado en capítulos anteriores.

- **Batería de pruebas del módulo PluginManager**

  - *Comprobación de la inicialización del PluginManager*

    El *'pm_logicLayer'* se encargará de la inicialización de la base de datos. Esta inicialización debe ser comprobada, ya que si esta falla no se podrá tener acceso a la información de localización de plugins, listas de procesado y caché de contenidos. El sistema es capaz de funcionar sin las listas de procesado y la caché de contenidos, pero es fundamental la información de registro de plugins.

| **Situación** | **Decisión** |
|---|---|
| Database Path corrupto | Callback a Xplico con 'priority' -1, y 'message' [ERROR_OPENING_DATABASE]. |
| Database Path correcto y Tabla Cache corrupta | Comprobaciones en caché desactivadas. Continua proceso. |
| Database Path correcto y Tabla Plugins corrupta | Callback a Xplico con 'priority' -1 y 'message' [ERROR_NO_PLUGIN_AVAILABLE]. |

Tabla 5.2: *Evaluación: Chequeo de la inicialización del PluginManager*

  - *Paso de petición por Xplico*

    Xplico debe construir una estructura de datos llamada XPLICO_ANALYSIS_REQUEST o XPLICO_TRAINING_REQUEST, para pasarla por parámetro a las funciones del PluginManager.

El PluginManager debe verificar que estas estructuras se encuentran bien construidas, ya que existen campos de datos que son imprescindibles y no pueden ir vacíos.

| **Situación** | **Decisión** |
|---|---|
| Alguno de los parámetros 'caseCategory', 'caseID', 'contentPath' Xplico los mete vacíos. | Callback a Xplico con 'priority' -1, y 'message' [ERROR_CORRUPT_REQUEST]. |
| Todos los campos están correctamente introducidos. | Continua proceso. |

Tabla 5.4: *Evaluación: Chequeo de la correcta petición de procesado de Xplico.*

- *Plugin Online*

Vamos a analizar las situaciones en que puede verse el PluginManager cuando el Plugin se encuentra Online y como actúa para obtener el mejor resultado.

| **Situación** | **Decisión** |
|---|---|
| El Plugin muere durante el procesamiento de un contenido. | Callback a Xplico con 'priority' -1 y 'message'[ERROR_PLUGIN_BREAK_DOWN]. |
| El Plugin stub está corrupto. | Callback a Xplico con 'priority' -1 y 'message'[ERROR_PLUGIN_INVOKE]. |
| La conexión SOAP con el Plugin falló | Callback a Xplico con 'priority' -1 y 'message'[ERROR_PLUGIN_CONNECTION]. |
| El Plugin no es capaz de procesar el contenido. | Callback a Xplico con 'priority' -1 y 'message'[mensaje personalizado] |

Tabla 5.6: *Evaluación: Chequeo de situaciones cuando Plugin está Online.*

- *Plugin Offline*

Vamos a analizar las situaciones en que puede verse el PluginManager cuando el Plugin se encuentra Offline y como actúa para obtener el mejor resultado.

| Situación | Decisión |
|---|---|
| Contenido a procesar en Caché. | Callback a Xplico con 'priority' y 'messages' de Caché. |
| Petición de procesado | Callback a Xplico con 'priority' -1 y 'message'[ERROR_PLUGIN_UNREACHABLE]. |

Tabla 5.8: *Evaluación: Chequeo de situaciones cuando Plugin está Offline.*

- *Chequeo de la correcta actualización de la caché de contenidos*

  Tal y como se ha diseñado el almacenamiento de contenidos en caché, se debe verificar que realmente se actualizan los resultados de los contenidos tras su procesado, tanto de análisis como en entrenamiento. Cuando mandamos a analizar un contenido y su resultado no se encuentra almacenado en caché, éste es procesado por un plugin y su respuesta es almacenada en caché. Cuando un contenido es entrenado, si su resultado no estaba almacenado aún en caché se añade directamente a la misma, pero si por el contrario ya existía una clasificación en caché para el mismo, este valor es modificado directamente. De este modo dispondremos de una caché siempre actualizada.

  Este funcionamiento podemos comprobarlo fácilmente a través del interfaz web mandando analizar un contenido, viendo su inclusión en caché y posteriormente entrenándolo con otro valor. Notaremos que el valor a parte de haberse actualizado en la sección de contenidos, también se habrá actualizado en caché.

- *Chequeo de la capacidad para procesar contenidos en paralelo*

  Tal y como se especificó en el capítulo de diseño, el PluginManager es capaz de lanzar procesamiento de contenidos asíncronamente, es decir, lanzar peticiones a plugins y que estos se procesen en paralelo. Para verificar su correcto funcionamiento se creó en el interfaz web desarrollado la funcionalidad de Analizar todos los contenidos.

- *Tiempos de ejecución*

  En las Tablas 5.10 y 5.12 se recogen los tiempos de ejecución del módulo cliente del PluginManager desde que Xplico llama a una de sus funciones de procesado de contenidos hasta un determinado punto (*breakpoint*) de la aplicación. Todas las mediciones se han realizado bajo una llamada de procesamiento de tipo *'Analyze'* al plugin *'spamassassin'*, a través del siguiente comando en consola:

```
time(./bin/pm.client/pluginManagerClient 2 'email.eml'
        '/opt/xplico/resources/email.eml' 'message/rfc822')
```

Del resultado de tal comando obtenemos tres tiempos: *'real'*, *'user'*, *'sys'*. De estos tres el valor de tiempo que encontramos en la tabla es el *'real'*.

| ***Breakpoint*** | **Tiempo de ejecución (sg)** |
|---|---|
| Después del chequeo de Cache de contenidos | 0.006s |
| Después del chequeo de Disponibilidad de plugins | 0.006s |
| Después de la serialización de la petición en un contenedor SOAP | 0.012s |
| Después de la emisión de petición de procesado | 0.013s |
| Después de la recepción de respuesta de plugin | 1.624s |
| Después de la deserialización del contenedor SOAP | 1.663s |
| Después de la actualización de Cache de contenidos | 1.683s |
| Después de la devolución de respuesta a Xplico | 1.691s |
| Ejecución total. Junto a la capa de adaptación de Xplico desarrollada. | 3.013s |

Tabla 5.10: *Evaluación: Tiempos de ejecución del PluginManager - Plugins remotos*

| ***Breakpoint*** | **Tiempo de ejecución (sg)** |
|---|---|
| Después del chequeo de Cache de contenidos | 0.006s |
| Después del chequeo de Disponibilidad de plugins | 0.006s |
| Después de la construcción de la petición | 0.006s |
| Después de la recepción de respuesta de plugin | 1,124s |
| Después de la actualización de Cache de contenidos | 1,156s |
| Después de la devolución de respuesta a Xplico | 1,161ss |
| Ejecución total. Junto a la capa de adaptación de Xplico desarrollada. | 3.006s |

Tabla 5.12: *Evaluación: Tiempos de ejecución del PluginManager - Plugins locales*

Ante estos resultados obtenidos, podemos concluir que como era de esperar, la ejecución local de plugins es más rápida que la remota. Aunque en estas pruebas su diferencia no es muy significativa, debido a que el tamaño y recursos demandados por el plugin analizador es reducida. De modo que en sistemas con Plugins de mayor tamaño esta diferencia de tiempos se hará realmente significativa.

- **Batería de pruebas de los Plugins de ejemplo**

  De entre los tres plugins de ejemplo desarrollados ya se ha comentado que únicamente el que tiene una funcionalidad acorde a su descripción es el *'Spamassassin'*, de modo que éste va a ser el único que evaluaremos en este apartado.

  Primero vamos a medir el tiempo de ejecución del Plugin para una petición de procesado de entrenamiento. Para ello se ha utilizado el siguiente comando en consola:

```
time(sudo ./run.as.local.sh 'training' 'caseCategory%%x' 'caseID%%x'
    'contentPriority%%3'
    'contentURI%%ftp://pluginmanager:pluginmanager@127.0.0.1:21/xplico
    /resources/email.eml'
    'contentHash%%185a758bbb29d14db0c767563171bb15' 'contentName%%email.eml'
    'contentType%%message rfc/822' 'contentDescription%%Descripcion'
    'contentLength%%12497' 'contentTimestamp%%11610848')
```

  El resultado obtenido tras su ejecución es el siguiente:

```
real    0m0.275s
user    0m0.140s
sys    0m0.120s
```

  Ahora analicemos el tiempo de ejecución del Plugin ante la petición de procesado de tipo análisis. Para ello se ha utilizado el siguiente comando en consola:

```
time(sudo ./run.as.local.sh 'analysis' 'caseCategory%%x' 'caseID%%x'
    'contentURI%%ftp://pluginmanager:pluginmanager@127.0.0.1:21/xplico
    /resources/email.eml'
    'contentHash%%185a758bbb29d14db0c767563171bb15' 'contentName%%email.eml'
    'contentType%%message rfc/822' 'contentLength%%12497'
    'contentTimestamp%%11610848')
```

El resultado obtenido tras su ejecución es el siguiente:

```
real    0m0.313s
user    0m0.190s
sys    0m0.120s
```

Comparando los datos medidos para cada uno de los procesos, podemos ver que el proceso de entrenamiento es algo más rápido que el de análisis. Esto es debido a que al entrenar lo único que se está haciendo es actualizar la cache del Plugin y su base de datos de contenidos, mientras que al analizar un contenido se tiene que llamar al software anti-spam antes de actualizar el resultado. Igualmente esta diferencia de tiempos entre los dos diferentes tipos de procesado disponibles no es muy significativa, pero como ya se comentó más arriba, es debido a que la lógica del Plugin *Spamassassin* es muy simple y no consume apenas recursos. Esa diferencia de tiempos se hará realmente notable cuando se desarrollen Plugins cuya lógica de procesado demande muchos recursos y tarde mucho tiempo en generar los resultados.

Nota: estas ejecuciones de procesado se han realizado sobre el contenido *email.eml*, el cual pesa 754 bytes.

Capítulo 6

# PRESUPUESTO DEL PROYECTO

En este capítulo se muestran justificados los costes globales de realización del presente Proyecto Final de Carrera. Estos costes asociados principalmente a gastos personales y de material quedan reflejados en las tablas presentadas en el documento de presupuesto adjuntado en la Figura 6.1.

- **Descripción del Proyecto**

  El desarrollo del proyecto a sido llevado a cabo entre mediados de octubre y mediados de julio del año 2011, haciendo un total de aproximadamente 9 meses de trabajo.

  En él han trabajado un Ingeniero de Telecomunicaciones como desarrollador principal y un Ingeniero Senior como Coordinador/Tutor.

  El cálculo de los costes totales ha sido obtenido a partir de las siguientes consideraciones:

  - Según el COIT (Colegio Oficial de Ingenieros de Telecomunicación), el Salario estipulado para un Ingeniero es de 26,14 €/hora. Suponiendo un total de 1.760 horas/año resultará un coste anual de 46.000 €.
  - Suponiendo que un Ingeniero Senior cobrará por encima de lo estipulado en el COIT, debido a que sus labores de coordinación así lo requieren, se asignará como referencia un salario de 30,68€/hora.

  Estos valores indicados son únicamente una referencia aproximada, que en función de la empresa en la que se desarrolle el proyecto variarán.

  En la Tabla 6.2 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas.

| **Fase** | **Tiempo (horas)** |
|---|---|
| Diseño | 130 |
| Implementación | 680 |
| Evaluación de la solución | 280 |
| Documentación | 230 |

Tabla 6.2: *Presupuesto del proyecto: fases del proyecto.*

**PRESUPUESTO DE PROYECTO**

1.- Autor:

Rubén Mena Escribano

2.- Departamento:

Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo: *"Módulo de distribución y pre-clasificación de contenidos para una plataforma de interceptación legal de comunicaciones"*
- Duración (meses): 9
- Tasa de costes Indirectos 20%

4.- Presupuesto total del Proyecto (valores en €):

5.- Desglose presupuestario (costes directos)

**PERSONAL**

| Apellidos y nombre | N.I.F. | Categoría | Dedicación (hombres mes) | Coste hombre mes *a)* | Coste Total (€) | Firma de conformidad |
|---|---|---|---|---|---|---|
| Urueña Pascual, Manuel | | Ingeniero Senior | 1 | 4.500,00 | 4.500,00 | |
| Mena Escribano, Rubén | | Ingeniero | 1 | 3.833,86 | 3.833,86 | |
| | | **Hombres mes:** | 2 | **Total** | 75.004,74 | |

a) Según el COIT (Colegio Oficial de Ingenieros de Telecomunicación), el Salario estipulado para un Ingeniero es de 26,14 €/hora. Suponiendo un total de 1.760 horas/año resultará un coste anual de 46.000 €. Supondremos que un Ingeniero Senior cobrará por encima de lo estipulado en el COIT, tomando 30,68 €/hora como referencia.

**EQUIPOS**

| Descripción | Coste (€) | Uso dedicado proyecto (%) | Dedicación (meses) | Periodo de depreciación | Amortización (€) *b)* |
|---|---|---|---|---|---|
| Ordenador (gama media) | 700,00 | 100 | 9 | 60 | 105,00 |
| SO Ubuntu | | 100 | 9 | 60 | |
| Licencias software | | 100 | 9 | 60 | |
| | | | | **Total** | 105,00 |

b) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

**SUBCONTRATACIÓN DE TAREAS**

| Descripción | Empresa | Amortización (€) |
|---|---|---|
| | | |
| | **Total** | |

**OTROS COSTES DIRECTOS DEL PROYECTO**

| Descripción | Empresa | Amortización (€) |
|---|---|---|
| | | |
| | **Total** | |

6.- Resumen de costes

| Factores | Costes Totales |
|---|---|
| Personal | 75.005 |
| Amortización | 105 |
| Subcontratación de tareas | |
| Costes de funcionamiento | |
| Costes Indirectos | 15.022 |
| Total | 90.132 |

Figura 6.1: Presupuesto del proyecto.

Capítulo 7

# CONCLUSIONES Y TRABAJOS FUTUROS

Como se introdujo al comienzo de la memoria, el principal objetivo del proyecto era conseguir un módulo de distribución y pre-clasificación de contenidos que fuese capaz de integrarse en una plataforma de interceptación legal de comunicaciones.

Para poder alcanzar este objetivo ha sido necesario un estudio previo del entorno de seguridad Europeo donde se ha visto la necesidad de un sistema de interceptación legal de contenidos de red, dando como resultado la idea de desarrollar tal módulo de distribución y pre-clasificación mencionado. Esta herramienta quedaría al servicio de los cuerpos de seguridad, facilitando la labor de clasificación final, ahorrando tiempo y optimizando recursos.

Tomada la decisión de embarcarnos en el proyecto, se ha llevado a cabo un estudio de diseño que analiza los diferentes medios, tecnologías y métodos que podrían hacer del software a desarrollar lo más óptimo, escalable y robusto posible. Para ello han sido necesarias varias reuniones de grupo donde se han debatido todos estos aspectos.

Su desarrollo ha tratado de ajustarse en todo momento a la firme idea de pre-clasificar contenidos de red, ya que la decisión de clasificación final debe ser siempre responsabilidad del operario de seguridad del sistema. Los contenidos analizados se enmarcan tanto en el ámbito único de la red (ej.: contenidos pedófilos, como imágenes) como en actividades ilícitas del mundo real (no virtual) que los criminales desempeñan utilizando internet como medio de comunicación.

Su correcto desarrollo se ha logrado gracias a la documentación que puede observarse en la bibliografía expuesta al final del documento, además de la colaboración del Tutor de proyecto.

En cuanto al desarrollo de la comunicación remota entre el PluginManager y los Plugins, ha sido de gran utilidad la página oficial de http://wso2.com/ la que en un principio se tomo como referencia en el descubrimiento del desarrollo de Servicios Web en C/C++, y de donde se han consultado varios artículos de documentación y desarrollo de *Webservices* usando Axis2. Posteriormente al conocimiento de Axis2 como proyecto de Servicios web más utilizado en la actualidad, se giraron las miras de desarrollo a la documentación oficial del Proyecto Axis de Apache, localizadas en las siguientes direcciones:

http://axis.apache.org/axis2/c/core/index.html

http://axis.apache.org/axis2/java/core/index.html

Para el desarrollo de la lógica interna del PluginManager, desarrollado en C/C++ ha sido de gran utilidad la documentación localizada en la web http://cplusplus.com/ y diversos foros de desarrollo.

Para la comprobación del correcto funcionamiento del software implementado se desarrollaron varios plugins de prueba, uno ubicado en local y dos en remoto para probar la comunicación vía *webservices*. Los Plugins implementados en local son llamados a través de una simple llamada a un script que configura ciertos parámetros y lanza el Plugin por consola, lo cual permite que el plugin pueda ser desarrollado en cualquier lenguaje de programación. Sin embargo para Plugins de ejecución remota, no se ha conseguido encontrar información que facilite el desarrollo de Plugins en un lenguaje de programación diferente a Java, quedando por el momento los desarrolladores de Plugins obligados a implementar la lógica de Plugins en éste lenguaje.

Desde un principio la idea inicial ha sido la de integrar el PluginManager con la herramienta Xplico, de tal modo que su implementación siempre se ha basado en la idea de intentar simplificar el proceso de integración, tratando de generar un modulo altamente integrable para conseguir la rápida aprobación del desarrollador de Xplico a la hora de llevar a cabo la integración.

Sin embargo esta integración necesitaría de la aprobación previa de su desarrollador principal, donde se expusiese la forma de llevar a cabo el proceso. Debido a la falta de tiempo se ha dejado el módulo en manos del tutor de proyecto el cual se encargará en un futuro de su integración. De modo que debido a la ausencia de integración, para poder demostrar gráficamente su correcto funcionamiento se ha desarrollado un interfaz web programado en CakePHP que junto a una pequeña clase programada como capa de adaptación con el PluginManager hacen de sustituto de Xplico.

De modo que los posibles trabajos futuros relacionados con el presente proyecto quedan expuestos a continuación.

- Como primer objetivo futuro queda pendiente reunirse con el desarrollador de la herramienta Xplico para negociar el modo en el que el PluginManager puede verse integrado para conseguir desplegar un sistema de interceptación legal de comunicaciones. Este trabajo requiere de la puesta de acuerdo de ambas partes, de ahí nuestro inicial interés comentado en

el capítulo anterior de tratar de desarrollar el PluginManager de una forma fácil y sencilla de cara a la integración, aunque eso haya llevado en ocasiones a complicar el desarrollo de su lógica interna.

- Como objetivos adyacentes se necesitaría comenzar a desarrollar Plugins de procesado de los contenidos de red más habituales, como podrían ser, el análisis de voz sobre IP (VoIP), análisis de imágenes, texto, etc. Me consta que en paralelo a este proyecto se encuentran varios compañeros desarrollando Plugins para verse inmediatamente a su finalización integrados con en la arquitectura del PluginManager. La detallada documentación proporcionada en esta memoria permitirá a los desarrolladores de Plugins conocer los mecanismos para desarrollar por si mismos Plugins de pre-clasificación sin más complicaciones que las que su propia lógica requiera, debido a que el sistema de implementación mostrado es muy simple. Se ha querido diseñar un modo que abstraiga lo máximo posible a los desarrolladores de plugins en su labor y los anime a continuar desarrollándolos.

- Otro aspecto a mejorar sería la investigación de un mecanismo para permitir generar código en el lado del Servidor de Web Services en un lenguaje diferente a Java, para permitir a los desarrolladores de Plugins optar por otros lenguajes de programación, ya que como se comentó anteriormente, actualmente no se proporcionan tales mecanismos y los desarrolladores se ven obligados a utilizar Java. Java es un lenguaje bastante común que como lenguaje inicial no está nada mal y creemos que su gran conocimiento por los desarrolladores no supondrá ningún problema para la expansión de implementación de Plugins. De todos modos sería conveniente buscar un camino para permitir elegir a los desarrolladores con el lenguaje que quieran programar ya que esto beneficiará en gran medida a la motivación para expandir el sistema.

- El sistema implementado cuenta con tecnologías que se encuentran en constante evolución, como servidores web, middlewares como CakePHP, que resulta muy aconsejable mantenerlos siempre en su última versión y realizar las modificaciones necesarias para soportarlos.

Para finalizar, a nivel personal debo decir que la realización del proyecto a sido una experiencia muy gratificante, debido a que he podido emplear todo lo aprendido y demostrar mi capacidad para llevar a cabo la realización de un proyecto de desarrollo software en el que se han visto incluidos tecnologías que nunca había visto antes y me han obligado a documentarme por mi

cuenta y pelearme con el problema planteado hasta hallar la solución. Me ha dotado de una experiencia, capacidad, e independencia en el trabajo fundamentales para poder pasar al siguiente nivel de preparación en mi vida.

Como conclusión se puede afirmar que el proyecto ha alcanzado sus objetivos inicialmente planteados.

# Bibliografía


[1] Apache axis2/c proyect (http://axis.apache.org/axis2/c/core/index.html). Ultimo acceso a 10 de Julio de 2011.

[2] Apache axis2/java proyect (http://axis.apache.org/axis2/java/core/index.html). Ultimo acceso a 10 de Julio de 2011.

[3] Cakephp official site (http://cakephp.org/). Ultimo acceso a 10 de Julio de 2011.

[4] Proyecto Indect (http://www.indect-project.eu/). Ultimo acceso a 10 de Julio de 2011.

[5] Gianluca Costa & Andrea De Franceschi. Xplico, network forensic analysis tool (http://www.xplico.org/). Ultimo acceso a 10 de Julio de 2011.

[6] Sistemas de interceptación ilegal de comunicaciones (www.wikipedia.com). Ultimo acceso a 10 de Julio de 2011.

[7] Sqlite software library (www.sqlite.org). Ultimo acceso a 10 de Julio de 2011.

[8] WSO2 (www.wso2.com). Ultimo acceso a 10 de Julio de 2011.

[9] Javier García de Jalón, José Ignacio Rodríguez, Rufino Goñi Alfonso Brazález, Patxi Funes y Rubén Rodríguez. Aprenda lenguaje ANSI C como si estuviera en Primero, Abril 1998. Ultimo acceso a 10 de Julio de 2011.

[10] Javier García de Jalón, José Ignacio Rodríguez, José María Sarriegui y Alfonso Brazález. Aprenda C++ como si estuviera en primero, Abril 1998. Ultimo acceso a 10 de Julio de 2011.

[11] C++ language site (www.cplusplus.com), 2000-2011. Ultimo acceso a 10 de Julio de 2011.

[12] Pedro Crespo. TEX/LATEX Manual, Junio 1999. Ultimo acceso a 10 de Julio de 2011.

[13] Walter Mora F. and Alex Borbón. Edición de textos científicos LATEX. Escuela de Matemáticas, Instituto Tecnológico de Costa Rica, 2009. Ultimo acceso a 10 de Julio de 2011.

[14] Ruchith Fernando. Hello world with apache axis2. Software Engineer WSO2 Inc, Mayo 2006. Ultimo acceso a 10 de Julio de 2011.

[15] Deepal Jayasingha. Writing your own services.xml for axis2 web services. Software Engineer WSO2 Inc, Julio 2007. Ultimo acceso a 10 de Julio de 2011.

[16] Nilesh D Kapadia. Embedding apache tomcat 7.0., 2010. Ultimo acceso a 10 de Julio de 2011.

[17] Damitha Kumarage. Understanding axis2/c threading model. Technical Lead WSO2 Inc, Enero 2009. Ultimo acceso a 10 de Julio de 2011.

[18] Lahiru Sandakith. Developing web services using apache axis2 eclipse plu-gins. Software Engineer WSO2 Inc, Junio 2007. Ultimo acceso a 10 de Julio de 2011.

[19] Eran Chinthaka. Debugging axis2 web services deployed within tomcat. Inc., Software Engineer WSO2, Julio 2008. Ultimo acceso a 10 de Julio de 2011.

# ANEXO A

# Instalación del proyecto

En este capítulo se va a exponer el aspecto que presenta el fichero localizado en el directorio raíz del proyecto, con nombre *install.sh* y que lleva a cabo la instalación de todos los paquetes necesarios para la correcta compilación y ejecución de todos los módulos del proyecto.

Su contenido es el siguiente:

```
#!/bin/bash

# This bash script installs all the packages that the
# PluginManager needs to be compiled, booted and be tested.

# -------- INSTALLATION AND CONFIGURATION SOME PACKAGES  --------

# openjdk-6-jdk - OpenJDK Development Kit (JDK).
apt-get install openjdk-6-jdk

# Apache Axis2/C - Apache web services engine. Development.
apt-get install libaxis2c-dev

rm -Rf ./lib/axis2c_src_1.6.0/
cd lib
tar xvfz  axis2c-src-1.6.0.tar.gz
cd  axis2c-src-1.6.0
./configure
make
make install
cd ../../

# libsqlite3-0 - SQLite 3 shared library.
apt-get install libsqlite3-dev
```

```
# sqlitebrowser - GUI editor for SQLite databases.
apt-get install sqlitebrowser

# vsftpd - lightweight, efficient FTP server written for security
apt-get install vsftpd

mkdir /opt/xplico
mkdir /opt/xplico/resources
chmod -R 777 /opt/xplico/resources
cp ./test/* /opt/xplico/resources
cp ./conf/ftp/vsftpd.conf /etc/
service vsftpd restart

# apache2 - Apache HTTP Server metapackage
apt-get install apache2

# Installing additional modules
apt-get install php5
apt-get install libapache2-mod-php5
apt-get install javascript-common
apt-get install php5-sqlite

# Activation of modules
a2enmod php5
a2enmod rewrite
apache2ctl -l

# Substituting 'Apache HTTP Server' Configuration files
cp ./conf/apache/ports.conf /etc/apache2/
cp ./conf/apache/000-default /etc/apache2/sites-enabled/
```

```
# Mount GUI on 'Apache HTTP Server'
cp -r ./gui/pluginManager/ /var/www/
chmod -R 777 /var/www/pluginManager

# Restart 'Apache HTTP Server'
/etc/init.d/apache2 restart

# Extract hashlibpp
rm -Rf ./lib/hashlibpp_0_3_2/
cd lib
tar xvfz hashlibpp_0_3_2.tar.gz
cd ../

# Creating PluginManager directories
mkdir /usr/local/pluginManager
mkdir /usr/local/pluginManager/db
mkdir /usr/local/pluginManager/bin
mkdir /usr/local/pluginManager/log
mkdir /usr/local/pluginManager/config
chmod -R 777 /usr/local/pluginManager

cp ./db/pluginManager_database /usr/local/pluginManager/db/
cp ./bin/pm.client/pluginManagerClient.sh /usr/local/pluginManager/bin
cp ./conf/pm/config.txt /usr/local/pluginManager/config
chmod -R 777 /usr/local/pluginManager
```

# ANEXO B

# Compilación y ejecución del PluginManager

Como se ha comentado, el PluginManager cuenta con una parte Cliente (C/C++) que recibe peticiones de pre-clasificación y una parte Servidora (Java) que se encuentra en el lado del Plugin y actúa como capa de adaptación entre el PluginManager y los Plugins de procesado.
A continuación veremos los requisitos y la forma de compilar cada modulo del PluginManager.

## B.1. PluginManager módulo Cliente

Su código se encuentra implementado en C/C++, y para su desarrollo se han utilizado ciertos *'include files (.h)'* y librerías que deben ser especificadas a la hora de ser compilado. De modo que previa a la compilación del código debemos descargar e instalar los paquetes especificados en la Tabla B.2. Su código fuente se encuentra en: *'./src/pm.client/'*

Una vez instalados los paquetes indicados en la Tabla B.2 se podrá compilar el código satisfactoriamente. Para ello se deberá ubicar en el directorio raíz del código fuente especificado arriba y ejecutar el siguiente comando en consola:

```
export AXIS2C_HOME=/usr/local/axis2c;
g++ ./pm.client/*.cpp ./db/*.cpp ../lib/hashlibpp_0_3_2/src/*.cpp
-I /usr/local/include -I ../lib/hashlibpp_0_3_2/src
-I ${AXIS2C_HOME}/include/axis2-1.6.0
-I ${AXIS2C_HOME}/include/axis2-1.6.0/platforms
-I ${AXIS2C_HOME}/include/axis2-1.6.0/platforms/unix
-L /usr/local/lib -L ${AXIS2C_HOME}/lib
-l "axutil" -l "axis2_axiom" -l "axis2_parser" -l "axis2_engine" -l "sqlite3"
-o ../bin/pm.client/pluginManagerClient -w
```

En el directorio *'./src/pm.client/'* se proporciona un *script* de nombre *'build.sh'* que compila el código a través de los comandos citados.

La ejecución de éste módulo requiere de la configuración de la siguiente variable de entorno:

```
export LD_LIBRARY_PATH=${AXIS2C_HOME}/lib:$LD_LIBRARY_PATH
```

En el directorio *'./bin/pm.client/'* se proporciona un *script* de nombre *'pluginManager-Client.sh'* que arranca éste módulo.

| **Paquete** | **Descripción** |
|---|---|
| axis2c-src-1.6.0.tar.gz | Proyecto Apache Axis2 para el tratamiento con Web-Services en C. Podemos obtenerlo de la página oficial de Apache, o en *'./lib/'* Para evitar problemas con la arquitectura de la máquina en la que trabajemos, se aconseja descargar el código fuente de la misma en su última versión y compilarlo e instalarlo manualmente.<br>1. `./configure`<br>2. `make`<br>3. `sudo make install`<br>Por defecto la instalación almacena los *'include files'* en: *'usr/local/axis2c/include/'*, mientras que las librerías en: *'usr/local/axis2c/lib'*. |
| hashlibpp_0_3_2.zip | Paquete empleado para el cálculo de códigos Hash de contenidos de red. Se encuentra almacenado en *'./lib/'*. |

Tabla B.2: *Paquetes C/C++ necesarios en el desarrollo del PluginManager.*

## B.2. PluginManager módulo Servidor

Igualmente, la parte Servidora del PluginManager necesita de unas librerías específicas para poder ser compilado. Éstas pertenecen al paquete de desarrollo de Axis2 para Java (*'axis2java-1.5.3-bin'*), el cual se puede obtener de la página oficial de Apache o del directorio *'./lib/'*.

Su compilación se lleva a cabo por los desarrolladores de Plugins, ya que la capa de adaptación del PluginManager proporcionada se encuentra dentro del paquete de desarrollo de plugins. Su compilación queda detallada en el Anexo(C).

Su ejecución se puede llevar a cabo a través de Webservices, o a través de un script de nombre *'run.as.local.sh'* localizado en el directorio raíz del código fuente para desarrolladores de plugins.

**ANEXO C**

# Desarrollo y despliegue de Plugins Remotos en Tomcat 7

En este Anexo se muestran los pasos que debe seguir un desarrollador de Plugins para llevar a cabo la implementación y posterior despliegue de un Plugin en un Apache Tomcat 7.

- **Paso I: Descomprimir código fuente para Plugins *'pg_ source_ code.tar.gz'***

  Almacenado en: *'./src/pm.server.plugins/pg_ source_ code.tar.gz'*. Una vez descomprimido podemos observar la siguiente estructura de directorios extraídos:

```
|-- bin
|   ‘-- META-INF
|       ‘-- services.xml
|-- lib
|   ‘-- axiom-api-1.2.10.jar, axis2-kernel-1.5.3.jar
‘-- src
    ‘-- org
        ‘-- xplico
            |-- plugin
            |   |-- XplicoPlugin_interface.java
            |   ‘-- XplicoPlugin_main.java
            ‘-- pluginmanager
                |-- adaptionLayer
                |   |-- local
                |   |   ‘-- LocalPluginAdaptionLayer.java
                |   ‘-- remote
                |       |-- PgCallbackHandler.java
                |       |-- PgMessageReceiverInOut.java
                |       ‘-- PgSkeleton.java
                ‘-- common
                    |-- AnalysisRequest.java
                    |-- PluginResponse.java
                    ‘-- TrainingRequest.java
```

Cada uno de estos directorios contiene varios ficheros y/o subdirectorios, de los cuales los desarrolladores de Plugins únicamente deben modificar los localizados en:

*./src/org/xplico/plugin*

*./resources*

En la Tabla C.2 se comenta el contenido de cada uno de los directorios del árbol presentado.

| **Directorio** | **Descripción del contenido** |
|---|---|
| bin/ | Contendrá los '.class' una vez compilado el código fuente. |
| bin/META-INF/ | Recursos necesarios para la creación de un Plugin Remoto. |
| lib/ | Contiene librerías '.jar' necesarias para compilar el código. |
| src/ | Código fuente para el desarrollo de Plugins. Compuesto por dos paquetes:<br>*/org/xplico/pluginmanager*<br>*/org/xplico/plugin* |
| src/org/xplico/pluginmanager/ | Código fuente *'PluginManager-server'*, que representa la capa de adaptación entre el PluginManager-client y la lógica de procesado de contenidos del Plugin.<br>Es proporcionado por el PluginManager y los desarrolladores de Plugins no lo deben modificar.<br>Se encarga de deserializar peticiones, adaptar la información para que la lógica del Plugin sea capaz de entenderla, y serializar la respuesta generada por el plugin para contestar al PluginManager-client. |
| src/org/xplico/plugin/ | Código fuente que los desarrolladores de Plugins deberán utilizar para implementar la lógica interna del Plugin. Contendrá fundamentalmente dos clases java, un interfaz que contiene las funciones necesarias de procesado y una clase *'main'* que implementará dicho interfaz. |

Tabla C.2: *Anexo: Desarrolladores de Plugins. Código fuente para Plugins.*

- **Paso II: Implementación de la lógica de procesado y compilación**

  El directorio en el que se deben almacenar aquellas clases que implementen la lógica de procesado de un plugin es: *'./src/org/xplico/plugin'*

  Como se vio en la Tabla C.2, este directorio contiene de entrada dos clases, un interfaz de implementación y una clase *main* con la que los desarrolladores comenzarán la implementación. Los métodos declarados en el interfaz *'XplicoPlugin_interface'* y que el desarrollador debe comenzar implementando en la clase *'XplicoPlugin_main'* son los siguientes:

```
 /**********************************************************************
* Function used to get the name of this Plugin.
* @return a String that represents the name of this Plugin.
**********************************************************************/
public String getPluginName ();

/**********************************************************************
* Will be used by the PluginManager-server in order to process the
* PluginManager-client Request of 'Analyze' type .
* @param analisisRequest, Request of 'Analyze' type from PluginManager.
* @return 'PluginResponse' object, that represents plugin response.
**********************************************************************/
public PluginResponse processAnalyzeRequest (AnalysisRequest analysisReq)
   throws MalformedURLException;

/**********************************************************************
* Will be used by the PluginManager-server in order to process the
* PluginManager-client Request of 'Train' type.
* @param trainingRequest, Request of 'Train' type from PluginManager.
* @return 'PluginResponse' object, that represents plugin response
**********************************************************************/
public PluginResponse processTrainRequest (TrainingRequest trainingReq)
   throws MalformedURLException;
```

Se podrán utilizar más clases adicionales para complementar la lógica de procesado del Plugin, las cuales deberán almacenarse en éste mismo directorio: *'./src/org/xplico/plugin'*.

Según se va generando código lo más apropiado es ir compilándolo para ir depurando los errores que se vayan cometiendo. El directorio destino de compilación en el que se ubicarán las clases generadas será: *'./bin'*.

Para compilar el código son necesarias dos librerías de Axis2, que podemos localizar dentro del directorio *'./lib'*. Utilizar el iguiente comando en consola una vez situados en el directorio raíz de descompresión:

```
javac -classpath ./lib/axiom-api-1.2.10.jar:./lib/axis2-kernel-1.5.3.jar
./src/org/xplico/plugin/*.java ./src/org/xplico/pluginmanager/*.java -d ./bin/
```

- **Paso IV: Empaquetado del plugin (servicio - aar file)**

  Una vez terminada la implementación se procederá a la creación del servicio (plugin). En *Web-Services* los servicios se construyen empaquetando el código y ciertos ficheros de configuración en un archivo *jar* de extensión *'.aar'*.

  1. *Configuración del fichero 'services.xml' e inclusión en el '.aar'*

     Este fichero se encuentra almacenado en la carpeta *'./bin/META-INF'* y es un fichero de configuración que utilizará el servidor en el que despleguemos el servicio para poder ser llamado correctamente. Entre otras cosas en él se especificará el nombre del servicio (es decir, el del Plugin en cuestión), que operaciones están disponibles (en nuestro caso, serán la de Análisis y Entrenamiento), y demás parámetros que son irrelevantes para los desarrolladores de Plugins.

     De este fichero lo único que un desarrollador de Plugins debe modificar es el parámetro *'name'* de la etiqueta *'service'*, en el que deberá poner el nombre del Plugin.

  2. *Empaquetar el código en un '.aar'*

     Para generar este paquete debemos situarnos en el directorio *'./bin'* y usar el siguiente comando:

     ```
     jar -cvf nombre.plugin.aar ./* ./META-INF/*
     ```

     Esto generará el paquete *nombre.plugin.aar* dentro del directorio *'./bin'*.

3. *Aspecto final del Plugin empaquetado*

```
.
|-- lib
|-- META-INF
|   |-- MANIFEST.MF
|   `-- services.xml
`-- org
        `-- xplico
            |-- plugin
            |   |-- XplicoPlugin_interface.class
            |   `-- XplicoPlugin_main.class
            `-- pluginmanager
            |-- adaptionLayer
            |   |-- local
            |   |   `-- LocalPluginAdaptionLayer.class
            |   `-- remote
            |       |-- PgCallbackHandler.class
            |       |-- PgMessageReceiverInOut.class
            |       `-- PgSkeleton.class
            `-- common
           |-- AnalysisRequest.class
           |-- PluginResponse.class
           `-- TrainingRequest.class
```

- **Paso V: Desplegar plugin (servicio) en un Apache Tomcat**

  Una vez instalado el Tomcat en su última versión como se especifica en el ANEXO(D), se procederá a mover el Plugin ('.aar' generado) al Servidor. El directorio al que debe ser movido se encuentra dentro del directorio de instalación de Tomcat (con la funcionalidad Axis2 instalada). Una vez situados en el directorio raíz del Servidor la ruta a la que mover el fichero será:

  *'./webapps/axis2/WEB-INF/services/'*

- **Paso VI: Comprobación del despliegue**

  Para comprobar que el Plugin ha sido desplegado correctamente, se tiene que acceder al Servidor Tomcat desde un navegador a través de la siguiente dirección URL (suponiendo que el Servidor se encuentra desplegado en la misma maquina en la que estamos trabajando):

  *'http://localhost:8080/axis2/'*

## ANEXO D

# Instalación y configuración Apache Tomcat 7

- **Instalación**

  Para el despliegue de los Plugins utilizaremos la última versión de Apache Tomcat, actualmente la v.7. Podemos descargarlo de la página oficial de Apache:

  http://tomcat.apache.org/download-70.cgi

  Una vez descargado extraemos su contenido en el directorio en el que queramos alojarlo.

```
~> tar xvzf /tmp/apache-tomcat-7.0.4.tar.gz
```

  Para poder arrancar Tomcat correctamente es necesaria una previa configuración del entorno. Debemos añadir al PATH los directorios donde se encuentran instalados Java y Tomcat.

```
# Java JDK Home
export JAVA_HOME=/usr/lib/jvm/java-6-openjdk
PATH=$JAVA_HOME/bin:$PATH

# Tomcat Home
export CATALINA_HOME=/home/userAccount/Tomcat7
PATH=$CATALINA_HOME/bin:$PATH
```

  Analizando la estructura de directorios del tomcat descomprimido anteriormente, vemos que hay una carpeta con nombre *'webapps'*, que será donde se alojaran las aplicaciones web que queremos que Tomcat despliegue.

  Por último, para arrancar Tomcat y que despliegue todas las aplicaciones web alojadas en el directorio especificado y detenerlo se utilizarán unos scripts localizados en el directorio *'/bin'* de su instalación.

```
./startup.sh
./shutdown.sh
```

Para verificar el correcto despliegue del servidor podemos comprobarlo accediendo al mismo a través del navegador web: *'http://localhost:8080'*

- **Añadiendo funcionalidad Axis2 a Tomcat**

  Para permitir que Tomcat sea capaz de tratar con Servicios Web, es necesario desplegar un modulo adicional de Axis2 que aporta la funcionalidad necesaria para montar sobre él aplicaciones web que tratan con servicios web.

  Para ello el proyecto Axis2 de Apache a desarrollado un modulo empaquetado en un fichero de extensión *'.war'* que al situarlo dentro del directorio 'webapps' de Tomcat despliega una aplicación web que monta todos los servicios web que se encuentran dentro de su directorio *'/axis2/WEB-INF/services/'*.

  Este paquete *'axis2.war'* podemos localizarlo dentro del directorio *'./lib'* del código fuente de desarrollo de Plugins o en la página oficial de Apache.

- **Tomcat Embebido: desplegando Tomcat a través de un JAR (Java Class Loader)**

  Como se ha visto en la anterior sección para utilizar Tomcat debemos instalarlo y configurarlo para añadir la funcionalidad Axis2. Con el objetivo de simplificar el proceso y automatizarlo para usar un Apache Tomcat en cualquier máquina Linux sin necesidad de instalarlo, se ha desarrollado una clase Java que monta dinámicamente en temporal el Servidor Apache. Para ello se ha utilizado un paquete específico llamado *'tomcat-embed-core-7.0.2-sources.jar'* que se obtuvo de la página oficial de Apache.

  Junto a estas clases se utilizó el paquete *'axis2.1.5.war'* que proporciona toda la funcionalidad de axis al Tomcat embebido.

  Como resultado se obtuvo un paquete *'.jar'* que con su simple ejecución despliega un servidor Tomcat7 con el conjunto de servicios incluidos en la aplicación Axis2 embebida. Este paquete podemos localizarlo en el directorio *'./bin/pm.server.plugins/'* con el nombre de *'plugins.tomcat.embedded.jar'*, el cual se encuentra por defecto con los dos plugins que se han desarrollado para las pruebas del proyecto (spamassassin e imageAnalyzer).

  Para ejecutarlo basta con llamar al siguiente comando desde el directorio raíz de descompresión:

```
java -jar ./bin/pm.server.plugins/plugins.tomcat.embedded.jar
```

**ANEXO E**

# Instalación y configuración HTTPD Apache2 en Ubuntu/Debian

- **Instalación-Reinstalación**

  Antes de instalar el servidor Apache2 comprobamos si ya se encuentra instalado y en caso de estarlo lo eliminaremos para asegurarnos que al reinstalarlo obtendremos la última versión y con la configuración por defecto del mismo.

```
sudo apt-get remove --purge apache2
sudo apt-get install apache2
```

- **Configuración del endpoint en el que escuchará el servidor**

  Para ello modificaremos la directiva *'Listen'* del fichero de configuración *'Ports.conf'*. Para realizar las pruebas de desarrollo se configuró la dirección de localhost en el puerto 80 para comunicarnos con el servidor Apache que alojará el front-end cakePHP.

```
sudo gedit /etc/apache2/ports.conf
Listen 127.0.0.1:80
```

- **Instalación de Plugins**

  Para que el servidor apache2 pueda desplegar correctamente el GUI cakePHP desarrollado, debemos instalar una serie de plugins adicionales que le doten de toda la funcionalidad necesaria. Estos módulos son:

```
sudo apt-get install php5
sudo apt-get install libapache2-mod-php5
sudo apt-get install javascript-common
sudo apt-get install php5-sqlite
sudo a2enmod php5
```

- **Activación de módulos**

```
sudo apache2ctl -l
sudo a2enmod rewrite
```

- **Reiniciar Servidor**

  Cada vez que un fichero de configuración es modificado debemos aplicar sus cambios, para ello tenemos que reiniciar el servidor.

```
sudo /etc/init.d/apache2 restart
```

  En caso de que este comando no funcione, podemos forzar su reinicio con:

```
/etc/init.d/apache2 force-reload
```

Por defecto el *'DocumentRoot'* de donde Apache lee los archivos que componen cada elemento a desplegar apunta al directorio *'/var/www/'*. De modo que, será en este directorio donde se deben alojar las aplicaciones web que queremos hacer públicas en la dirección configurada previamente.

**ANEXO F**

# Instalación y configuración Servidor FTP en Ubuntu/Debian

- **Instalación-Reinstalación**

  Antes de instalar el servidor FTP en la máquina, debemos comprobar que no se encuentra ya instalado. En caso de estar instalado podemos pasar directamente a la configuración del mismo, pero si hacemos esto debemos asegurarnos que todos los ficheros configurables de importancia quedan reconfigurados, ya que si no lo hacemos correctamente la antigua configuración puede hacer que no funcione correctamente el despliegue. Para asegurarnos que partimos de una configuración por defecto lo mejor será reinstalarlo. Para ello eliminamos el modulo entero, asegurándonos de remover todos los ficheros de configuración antiguos y lo reinstalamos. En caso de no encontrarse instalado ya, simplemente lo instalamos.

  ```
  sudo apt-get remove --purge vsftpd
  sudo apt-get install vsftpd
  ```

- **Comprobación instalación**

  Una vez instalado debemos comprobar si se ha desplegado correctamente. Para ello consultaremos si el demonio *'vsftpd'* se encuentra corriendo.

  ```
  ps aux | grep vsftpd
  ```

- **Comprobación acceso al servidor**

  Llegados a este punto trataremos de acceder al servidor utilizando la cuenta de usuario de Ubuntu desde la que ha sido instalado.

  ```
  ftp 127.0.0.1
  ```

  Una vez obtenido el acceso observamos que el *root* del servidor apunta al directorio de usuario *'/home/userAccount/'*, el cual debe ser modificado para que apunte a *'/opt'*. Este puede ser modificado a través del comando *'proftpd'* o configurando el fichero *'vsftpd.conf'* que veremos a continuación.

- **Configuración**

  Para configurarlo deberemos editar el fichero *'vsftpd.conf'* donde deben dejarse tal y como están todas las directivas a excepción de las mostradas a continuación, y en caso de no existir introducirlas:

```
sudo gedit /etc/vsftpd.conf&
anonymous_enable=NO # in order to do more secure the server.
write_enable=YES
local_umask=022
ftpd_banner=Welcome Xplico FTPserver
local_enable=YES
pam_service_name=vsftpd
userlist_enable=YES
listen=YES
tcp_wrappers=YES
local_root=/opt
```

- **Reiniciar Servidor**

  Cada vez que un fichero de configuración es modificado debemos aplicar sus cambios, para ello tenemos que reiniciar el servidor.

```
sudo service vsftpd restart
```